ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

SVILUPPO DI APPLICAZIONI MOBILE PLATFORM-INDEPENDENT MEDIANTE TECNOLOGIE WEB - IL FRAMEWORK PHONEGAP

Elaborata nel corso di: Sistemi Operativi

Tesi di Laurea di:
ALESSANDRO ZOFFOLI

Relatore:
Prof. ALESSANDRO RICCI

ANNO ACCADEMICO 2010–2011
SESSIONE III

PAROLE CHIAVE

Mobile

Platform-Independent

Web

Javascript

HTML5

Alla mia famiglia.

Indice

Introduzione

La nascita della programmazione per dispositivi mobile risale a circa 10 anni fa, ma è soltanto a partire degli ultimi 3-4 anni che questa ha registrato una vera e propria crescita esponenziale. Le cause di questa esplosione sono da ricercare principalmente nella nascita ed evoluzione di un nuovo e rivoluzionario mezzo di comunicazione: lo smartphone. In realtà telefoni cellulari "smart" erano già in circolazione da qualche tempo, ma erano per lo più considerati come uno strumento di nicchia, riservati a professionisti. Le funzioni di tali device, oltre alle classiche funzioni di un telefono, consistevano principalmente nella lettura di email e nella consultazione di pagine web, tramite semplici browser che riuscivano a visualizzare semplice testo, link e immagini.

L'inizio di questo rivoluzionario cambiamento può essere attribuito all'uscita del primo iPhone, nel 2008. Il modo di utilizzare il telefono cambiò radicalmente e il cellulare divenne ben presto uno strumento non solo per comunicare, ma per ascoltare musica, navigare in rete, giocare, ... Parallelamente ad Apple, nel mercato mobile si inserì anche Google, con il suo sistema operativo Android e, nel corso dei mesi e degli anni, numerose altre aziende hanno proposto i loro sistemi operativi mobile.

Questa tesi si propone di redigere un quadro generale riguardante lo stato dell'arte della programmazione mobile cross-platform, introducendo un nuovo possibile approccio alternativo alla programmazione nativa. Nel primo capitolo verrà descritto lo stato dell'arte della programmazione nativa, che mette in evidenza come si ad oggi difficile per uno sviluppatore essere sempre aggiornato e preparato su ogni fronte e su ogni piattaforma specifica. Nel secondo capitolo viene brevemente introdotto il framework PhoneGap, che sarà ampiamente descritto ed analizzato. Il terzo capitolo descrive invece alcune tra le più note tecniche di mobile web developing, tecniche che sono ampiamente utilizzate sia nella creazione di applicazioni web, sia nella

creazione di applicazioni mobile cross-platform. Il quarto capitolo entra invece nel dettaglio e descrive le potenzialità e i principi di funzionamento del framework PhoneGap, portando anche numerosi esempi di codice, presi direttamente dalla documentazione ufficiale. Nel quinto capitolo si dimostrerà come sia possibile integrare le tecnologie e le tecniche descritte nei capitoli precedenti per progettare ed implementare una applicazione scelta come caso di studio. Il sesto capitolo contiene alcune riflessioni critiche ed introduce alcuni possibili sviluppi futuri, provenienti dalla ricerca. Il settimo capitolo contiene infine delle osservazioni conclusive.

Capitolo 1

Quadro generale e stato dell'arte

Negli ultimi anni il livello di diffusione dei dispositivi mobili nella vita di tutti i giorni ha raggiunto un livello tale che gli sviluppatori e le aziende che lavorano nel settore informatico hanno dovuto investire e reinvestire capitale umano e finanziario per acquisire e ricercare le competenze necessarie per offrire agli utenti una nuova user-experience che si adatti alle nuove tendenze.

La differenziazione dell'esperienza utente offerta dai vari produttori e dalle varie piattaforme diventa quindi ciò che caratterizza e distingue un sistema operativo mobile da un altro. Ogni sistema operativo ha le sue peculiarità, i suoi pattern di utilizzo ed è caratterizzato da una interfaccia grafica che può essere o meno personalizzabile dall'utente. Uno sviluppatore o un'azienda che vuole quindi cimentarsi nella creazione e sviluppo di applicazioni mobile non può quindi non confrontarsi con questa realtà estremamente frammentata relativa ai diversi sistemi operativi su cui le applicazioni dovranno girare.

Diverse sono le scelte che possono essere intraprese ed in generale non c'è una scelta migliore delle altre. Ciò che lo sviluppatore o l'azienda deve fare prima di cominciare lo sviluppo vero e proprio di una applicazione sarà quindi una pianificazione e una scelta riguardante le tecnologie con cui questa verrà implementata. Alcune possibili scelte sono le seguenti:

1. Viene scelto un SO mobile target e viene implementata l'applicazione. In seguito, dopo aver valutato il riscontro ricevuto dagli utenti, si può

decidere se proseguire l'implementazione anche per gli altri SO mobile. Tale approccio è detto platform-specific.

2. Viene utilizzato un approccio cross-platform, cercando di utilizzare tecnologie che permettono di implementare l'applicazione una sola volta e di offrire l'applicazione finita ad un numero più o meno ampio di SO target.

Ad oggi, la scelta predominante è stata sicuramente la prima. La flessibilità e le possibilità che la programmazione platform-specific offre agli sviluppatori non potrà mai essere raggiunta da soluzioni cross-paltform. D'altro canto, la varietà di piattaforme presenti, ognuna con le sue peculiarità e tratti distintivi, e l'universo eterogeneo e diversificato degli ambienti di sviluppo complica non poco la vita allo sviluppatore. Per cercare di semplificare lo sviluppo di applicazioni mobile sta prendendo sempre più piede la seconda scelta, che permette allo sviluppatore di implementare l'applicazione una sola volta per tutte le piattaforme. Tale approccio non può però essere utilizzato per ogni tipo di applicazione.

1.1 Tipologie di applicazioni mobile

L'universo delle applicazioni presenti negli store dei vari SO mobile è tutt'altro che omogeneo. Una possibile classificazione delle applicazioni presenti in base all'utilizzo che gli utenti ne fanno è la seguente:

- Social
- Intrattenimento
- Comunicazione
- Giochi
- Utility
- ...

Le applicazioni classificate come "social" sono tra le più diffuse e utilizzate dagli utenti. Sono relativamente di semplice utilizzo e hanno un'aspetto

che le identifica chiaramente. Qualche esempio di applicazioni classificate come "social": Google+, Twitter, Facebook, ecc...

Le applicazioni classificate come "intrattenimento" offrono servizi tra i più disparati. Il loro funzionamento solitamente si limita a compiere qualche lieve computazione e/o a comunicare con servizi web. Qualche esempio di applicazioni classificate come "intrattenimento": GoogleMaps, GoogleReader, Shazam, ecc... Queste due prime categorie si prestano piuttosto bene ai nuovi approcci di sviluppo cross-platform.

Le applicazioni classificate come "comunicazione" offrono servizi con cui l'utente può mettersi in contatto ed interagire con altre persone. Queste possono essere più o meno complesse, a secondo del servizio che offrono. Le più complete ad esempio possono offrire comunicazione audio/video, le più basilari offrono semplici servizi di messaggistica. Qualche esempio di applicazioni classificate come "comunicazione": GoogleTalk, Skype, WhatsApp, Viber, ecc...

Questa categoria risulta idonea allo sviluppo tramite tecnologie cross-platform solo nel caso in cui il servizio di comunicazione si appoggi su un web server a cui è delegato tutto il carico computazionale. Risulta inoltre di difficile realizzazione creare applicazioni di comunicazione real-time tramite questo tipo di approcci, in quanto utilizzano meccanismi che sono troppo legati alle specifiche piattaforme.

Per quanto riguarda i giochi, risulta evidente, almeno per quelli più avanzati, che lo sviluppo platform-specific è più che un requisito e deve inoltre essere affiancato dall'utilizzo di librerie grafiche che permettano l'utilizzo della GPU dei dispositivi. Il vero limite sta nel fatto che i vari dispositivi supportano in modo diverso tali librerie, per cui alcuni giochi potrebbero non girare affatto su diversi dispositivi che montano uno stesso SO. Qualche esempio di applicazioni classificate come "giochi": AngryBirds, FruitNinja, Let's Golf, ecc...

Le applicazioni classificate come "utility" comprendono tutte quelle applicazioni che possono estendere le funzionalità del dispositivo, e sono solitamente riservate agli utenti più avanzati. Qualche esempio di applicazioni classificate come "utility": setCpu, terminalEmulator, custom keyboard, ecc... Anche in questo caso risulta quindi difficile, se non impossibile l'utilizzo di aprocci cross-platform, in quanto queste applicazioni sono strettamente platform-specific e, in alcuni casi, device-specific.

1.2 Programmazione nativa

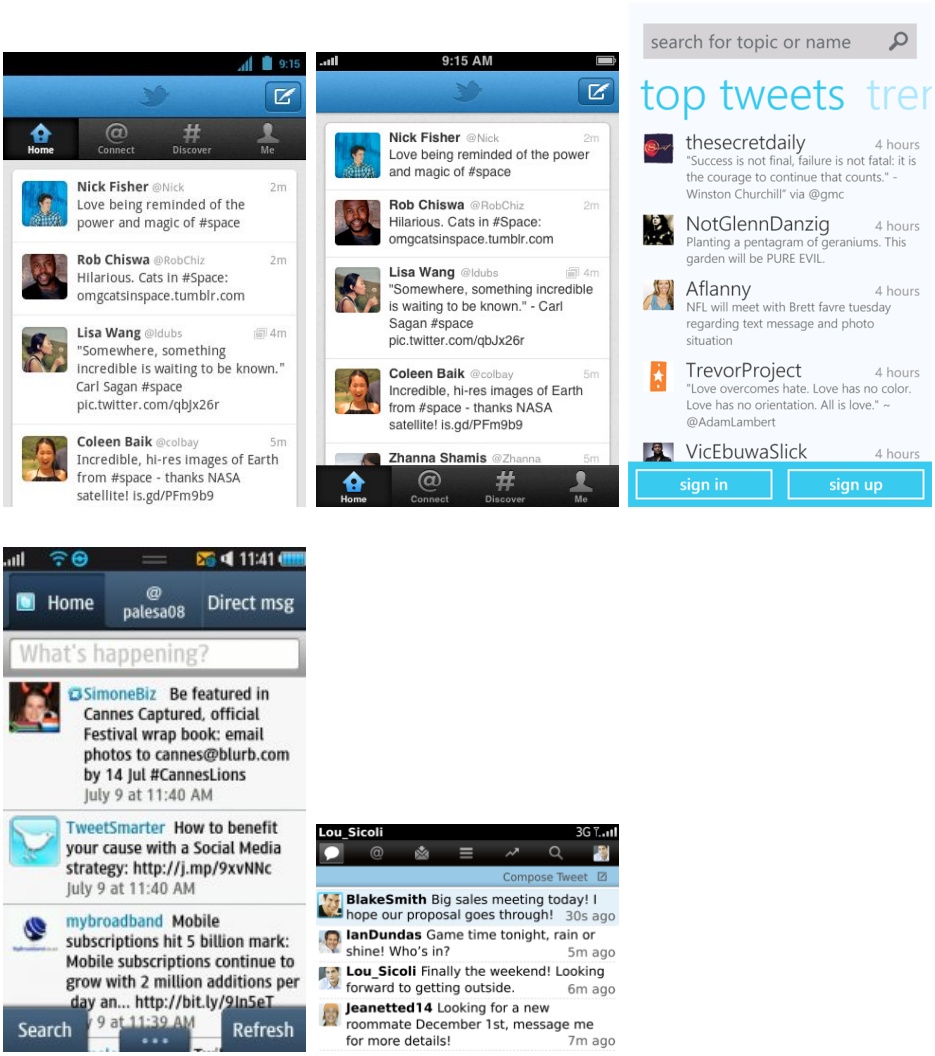
Con programmazione nativa si intende lo sviluppo di applicazioni utilizzando l'approccio platform-specific, che consiste nell'implementazione e nel successivo building dell'applicazione per ogni singola piattaforma. Lo svantaggio più grande di tale approccio risiede nel fatto che ogni piattaforma ha un suo ambiente di sviluppo, un suo sdk, un suo linguaggio caratteristico (solitamente un dialetto del C). Lo sviluppatore o il team di sviluppatori che sta dietro ad una applicazione dovrà quindi avere tutte le skill necessarie per poter sviluppare e portare l'applicazione su ogni piattaforma target. La seguente tabella riassume le maggiori piattaforme ad oggi presenti e le relative skill richieste.

Mobile OS	**Skill Set Required**
Apple iOS	C, Objective C
Google Android	Java (Dalvik VM)
RIM BlackBerry	Java (J2ME)
Symbian	C++, Python, HTML/CSS/JS
Windows Mobile	.NET
Windows 7 Phone	.NET
HP Palm webOS	HTML/CSS/JS
MeeGo	C, C++, HTML/CSS/JS
Samsung Bada	C++

Risulta subito evidente che il range di conoscenze e tecnologie è molto ampio ed eterogeneo. Inoltre sia gli ambienti di sviluppo che gli sdk vengono periodicamente aggiornati introducendo nuove funzionalità e nuove API di sistema. Risulta quindi molto difficile per uno sviluppatore essere sempre al passo con le ultime novità introdotte. A complicare ulteriormente il tutto c'è il fatto che gli OS mobile vengono periodicamente aggiornati, e non tutti i produttori di device forniscono l'upgrade per i loro dispositivi. È quindi possibile che una applicazione scritta per una versione di una certa piattaforma non sia compatibile con un'altra versione, della stessa piattaforma, più datata. Tali difficoltà non sono insormontabili soprattutto per le grandi aziende, che hanno team di sviluppatori dedicati ad una singola piattaforma, mentre sono un considerevole ostacolo per il singolo sviluppatore, che deve investire il suo tempo per restare aggiornato su tutti i fronti.

1.3 Un esempio: Twitter

Come esempio di applicazione nativa, Twitter può essere un'ottimo esempio. Seguono alcuni screenshots, per meglio evidenziare come una singola applicazione possa essere presentata diversamente a seconda della piattaforma su cui viene eseguita.



L'applicazione fornisce in tutte le sue diverse versioni le stesse funzioni, ovvero interagisce con i server di Twitter e permette all'utente di visualizzare/inviare dei tweet. Ciò che appare evidente è che ogni piattaforma

porta con sé delle peculiarità legate intrinsecamente all'esperienza utente che il SO mobile intende offrire. Questo discorso verrà approfondito nel seguito.

1.4 Framework mobile cross-platform

Nei paragrafi precedenti si è più volte sottolineato come sia difficile e frustrante essere sempre aggiornati e competenti nello sviluppo su più piattaforme. L'idea che ha portato alla nascita di framework cross-platform è partita dalla seguente osservazione: l'unica cosa che i vari OS mobile hanno in comune è che tutti possono accedere al web browser tramite codice nativo[13]. Utilizzando quindi linguaggi comuni e relativamente semplici per ogni sviluppatore, quali sono HTML5, CSS3 e Javascript, è possibile quindi creare applicazioni che si presentino in uguale maniera in tutte le diverse piattaforme su cui verrà distribuita l'applicazione.

Da queste osservazioni hanno iniziato a diffondersi diversi framework e librerie che semplificano non poco lo sviluppo su diverse piattaforme. Tali framework contengono parti di codice nativo, che permettono ad esempio l'utilizzo dei vari sensori presenti sul device, quali ad esempio la fotocamera, l'accelerometro, il gps, e wrappano l'accesso a questi sensori con librerie Javascript. Dall'interno di una webview è possibile quindi invocare codice nativo, specifico per ciascuna piattaforma, tramite le apposite librerie Javascript, che espongono una interfaccia comune per tutte le piattaforme. In sostanza:

- l'aspetto dell'applicazione viene modellato tramite HTML5 e CSS3
- il funzionamento viene esplicitato tramite codice Javascript, che, attraverso apposite librerie, può accedere ed utilizzare i sensori del device.

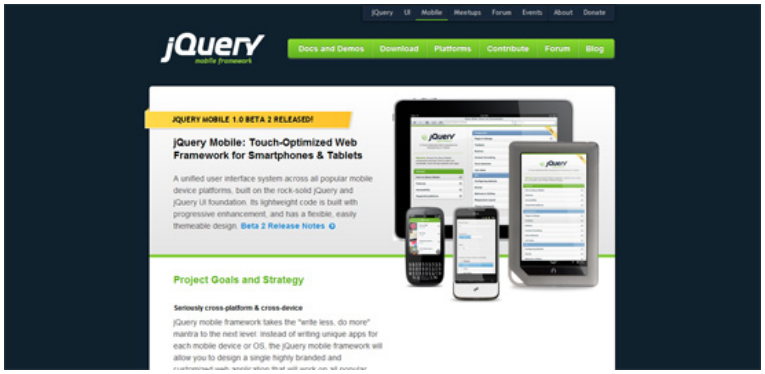
Ogni libreria, come detto in precedenza, ha una interfaccia standard, che non dipende dalla specifica piattaforma. Ciò permette agli sviluppatori di implementare l'applicazione una volta soltanto e di effettuare il building dell'applicazione per ogni piattaforma target, utilizzando le opportune librerie (che sono standard, ma dipendono dalla specifica piattaforma).

1.4.1 Framework più diffusi

Alcuni tra i framework più famosi ed utilizzati per realizzare mobile web application sono i seguenti:

jQuery Mobile

jQuery Mobile[1] è sicuramente il framework più famoso e popolare per lo sviluppo di mobile user interface e copre una vasto insieme di dispositivi. Il framework è relativamente leggero e offre un buon numero di elementi grafici a cui gli utenti mobile sono abituati, come ad esempio switch e slider. Tali elementi sono intrinsecamente legati al mondo mobile e touch, e lo differenziano da quello standard del "sito web fatto per i pc".



Sench Touch

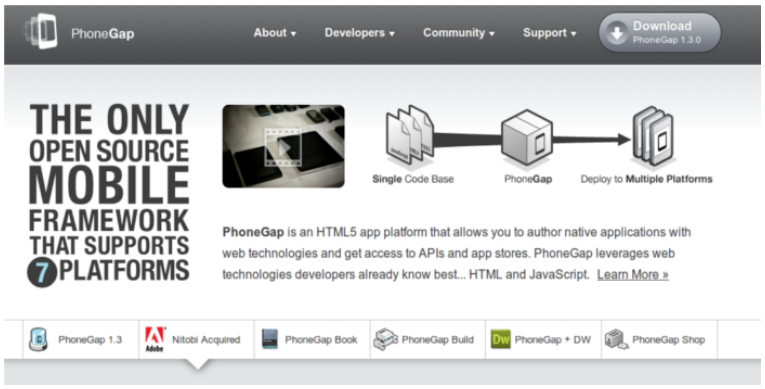
Sench Touch[2] è un framework molto potente e deriva da ExtJS. Anche questo framework mette a disposizione numerosi componenti che hanno poco da invidiare a quelli presenti nelle applicazioni native.

The-M-Project

The-M-Project[3] è un altro solido framework in Javascript che utilizza le nuove tecnologie portate da HTML5 per lo sviluppo di applicazioni web ottimizzate per i dispositivi mobile. Il framework è conforme al pattern Model-View-Controller e permette anche l'utilizzo offline da parte degli utenti.

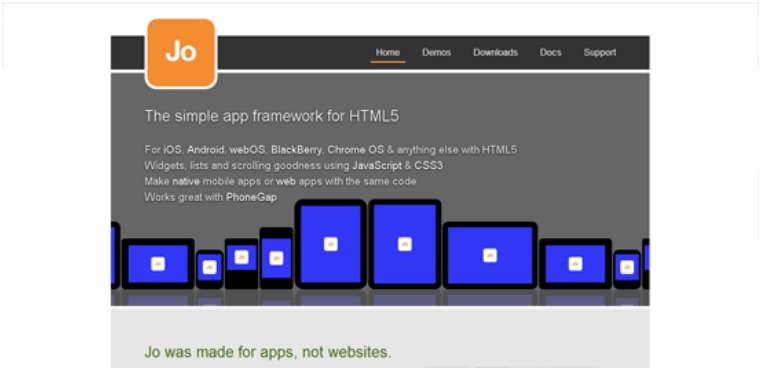
PhoneGap

[4] Verrà ampiamente trattato in seguito.



Jo

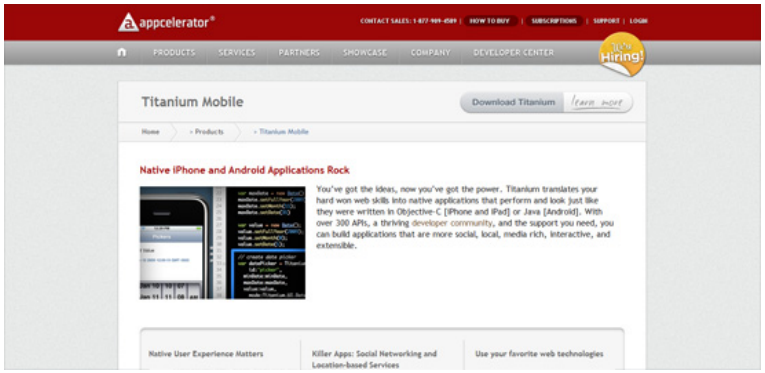
[5]



DHTMLX Touch

[6]

Titanium Mobile

[7] 

Non tutti questi framework in realtà offrono l'accesso ai sensori fisici dei device, e sono pensati per lo più per essere utilizzati all'interno di siti web ottimizzati per device mobili. Ciò non toglie che possano comunque essere utilizzati anche in tali applicazioni, ad esempio combinando il loro utilizzo con quello di altri framework, visto l'alta analogia fra l'ottimizzazione di una web application per dispositivi mobili, e l'approccio cross-platform precedentemente introdotto. Alcuni framework sono stati invece pensati e sviluppati per funzionare esclusivamente su device (come ad esempio Phonegap e Jo).

Capitolo 2

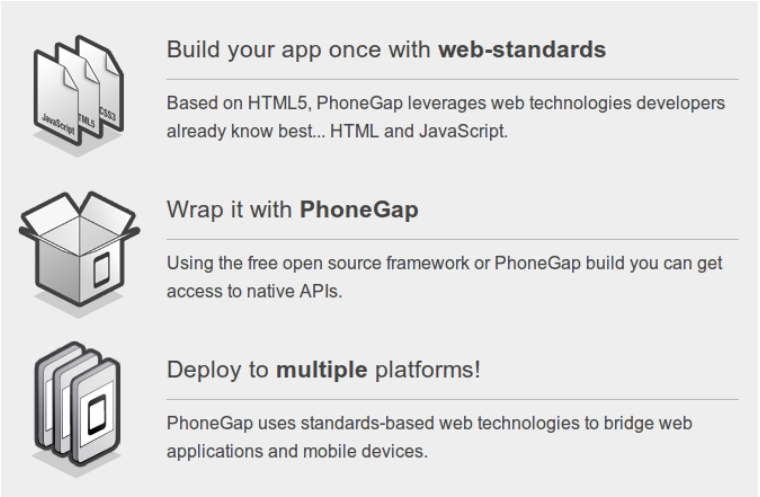
Il framework PhoneGap

PhoneGap è un framework scritto in Javascript che funge da bridge tra applicazioni mobile e applicazioni web. Il sito ufficiale ospita una comunità molto attiva e offre informazioni molto dettagliate sulle funzionalità e sui tool da utilizzare per riuscire rapidamente ad essere operativi.

La comunità di sviluppatori offre inoltre una serie di plugin, che è possibile aggiungere ai propri progetti, in continua espansione, con lo scopo di ridurre sempre di più il divario tra features native e feature web-styled.

2.1 Obbiettivi

Essenzialmente il suo obbiettivo è molto semplice: fornire agli sviluppatori, anche sprovvisti di conoscenze specifiche riguardanti le diverse piattaforme, uno strumento che permetta loro di scrivere applicazioni cross-platform nel modo più semplice e veloce possibile.



PhoneGap non è altro che un wrapper, un contenitore, che permette agli sviluppatori di incorporare le loro applicazioni web all'interno di applicazioni native di diverse piattaforme. Analogamente ai framework introdotti in precedenza, le applicazioni che utilizzeranno questo framework saranno scritte con HTML5, CSS3 e Javascript.

Il collegamento con le librerie native, specifiche di ogni piattaforma, sarà fatto tramite le API che mette a disposizione il framework. La filosofia che sta dietro a questo può essere riassunta dal famoso slogan "Write Once, Run Everywhere!".

2.2 Compatibilità

Il framework offre supporto per ogni piattaforma mobile degna di essere presa in considerazione. In particolare: Android, iOS, Bada, Symbian, WebOS, Blackberry. Non tutte le piattaforme sono però supportate allo stesso modo. La seguente tabella riassume lo stato dell'arte riguardo la compatibilità e l'accessibilità che il framework offre verso i sensori presenti nei diversi device, equipaggiati da diversi OS.

	iOS iPhone / iPhone 3G	iOS iPhone 3GS and newer	Android	OS 4.6-4.7	OS 5.x	OS 6.0+	WebOS	WP7	Symbian	Bada
ACCELEROMETER	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
CAMERA	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
COMPASS	✗	✓	✓	✗	✗	✗	✗	✓	✗	✓
CONTACTS	✓	✓	✓	✗	✓	✓	✗	✓	✓	✓
FILE	✓	✓	✓	✗	✓	✓	✗	✓	✗	✗
GEOLOCATION	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MEDIA	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗
NETWORK	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (ALERT)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (SOUND)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (VIBRATION)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STORAGE	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗

Nel seguito verrà illustrato come fare a capire a run-time se il device su cui sta girando una applicazione possiede o meno una particolare feature.

2.3 About PhoneGap

Il framework è un progetto open source ed è stato sviluppato originariamente dall'azienda canadese Nitobi, oltre che da una vasta comunità di sviluppatori, tra cui alcuni provenienti da IBM, RIM e Microsoft. Nell'ottobre 2011 Nitobi è stata ufficialmente acquisita da Adobe. PhoneGap, come precedentemente detto, è open source, e i sorgenti sono rilasciati con licenza Apache License 2.0.

Capitolo 3

Mobile web application developing

Per uno sviluppatore web sviluppare applicazioni con PhoneGap risulta abbastanza semplice ed intuitivo. Chiunque abbia nozioni di base di Javascript può infatti cimentarsi e creare semplici applicazioni. Per ottenere applicazioni equiparabili a quelle native è invece richiesto un livello di conoscenza delle tecnologie web un po' più elevato. Il semplice Javascript, utilizzato come tutti lo conoscono, ovvero come linguaggio di scripting, non può essere sufficiente.

Non si può pensare di modellare applicazioni, soprattutto se complesse, con un semplice linguaggio di scripting. In realtà Javascript non è solo un linguaggio di scripting, ma è un linguaggio object oriented, anche se con le sue limitazioni.

Per cominciare a sviluppare, il primo passo da compiere è quello della scelta e della configurazione dell'ambiente di sviluppo. Una volta scelta una piattaforma di partenza è possibile iniziare a testare le funzionalità messe a disposizione dal framework. In questo capitolo verranno illustrati una serie di approcci utili nella programmazione di applicazioni web per dispositivi mobili, indipendenti da PhoneGap.

3.1 HTML5

HTML5 è l'ultima "versione" di HTML, e il suo processo di specifica è tutt'ora in corso. HTML5 si propone di fornire un valore aggiunto alle

funzionalità classiche del classico HTML, inteso come semplice strumento di markup, ed è particolarmente importante in PhoneGap per la ricchezza e la qualità delle interfaccie di programmazione che mette a disposizione. Nel seguito verranno descritte alcune delle più importanti ed utilizzate funzionalità che dovranno essere prese in considerazione nello sviluppo di applicazioni mobile.

3.1.1 Media elements

Nelle applicazioni native è relativamente semplice, anche se un po "macchinoso" incorporare video/audio in una view. Nelle web application invece tale operazione risulta immediata e veloce.

I browser HTML5 compliant possono infatti incorporare audio e video nello stesso modo in cui incorporano immagini. I tag video e audio saranno quindi utilizzati analogamente a come viene utilizzato il tag img. Il seguente breve esempio mostra come inserire un video utilizzando HTML5.

```
<video id="sampleVideo" src="exVideo.mp4" controls></video>
```

I tag audio e video non sono altro che elementi DOM all'interno della pagina HTML. Tali tag permettono una elevata personalizzazione a livello di presentazione. Infatti è possibile ad esempio aggiungere i classici controlli che solitamente un riproduttore video/musicale deve avere, come una barra di riproduzione, controlli del volume, bottoni play/pause, e tanti altri. E' inoltre possibile disegnare controlli personalizzati. Tramite Javascript è possibile intercettare gli eventi che partono dai controlli ed eseguire il codice opportuno.

In rete esistono numerose librerie che offrono controlli personalizzati gia pronti, e ne permettono l'inclusione con poche righe di codice.

3.1.2 Canvas elements

HTML5 permette di disegnare dinamicamente elementi grafici mettendo a disposizione due diversi strumenti:

- Canvas Element, per grafiche di tipo bitmap;
- Scalable Vector Graphics, per grafiche vettoriali.

Siccome Android non supporta SVG, per creare grafiche dinamicamente sarà necessario usare gli elementi Canvas.

Gli elementi Canvas sono elementi HTML, ed è possibile quindi crearli e associare ad essi uno stile tramite CSS e Javascript. Per creare un Canvas, è sufficiente aggiungere alla pagina HTML il seguente codice:

```
<canvas id="myCanvas" width="100" height="100"></canvas>
```

Per associare uno stile al Canvas, è sufficiente aggiungere il seguente codice al foglio di stile:

```
#myCanvas {
  position: absolute;
  left: 280px;
  background: #ccf;
}
```

Una volta creato il Canvas è possibile disegnarci delle figure. La seguente funzione Javascript ad esempio permette di disegnare un cerchio.

```
function drawCircle() {
  var canvas = document.getElementById('myCanvas');
  var ctx = canvas.getContext('2d');
  var x = 25, y = 35, rd = 10;
  ctx.beginPath();
  ctx.arc(x, y, rd, 0, Math.PI * 2, false);
  ctx.closePath();
  ctx.fillStyle = «yellow»;
  ctx.fill();
}
```

I Canvas Elements possono essere molto utili ad esempio quando si vogliono creare dei semplici diagrammi o grafi, oppure delle applicazioni che permettono all'utente di disegnare.

3.1.3 Local storage

Una delle novità più interessanti ed utili di HTML5 è Local Storage, che permette di far memorizzare permanentemente dei dati all'applicazione. Il meccanismo su cui si basa è quello di una struttura chiave-valore ed il suo utilizzo è semplicissimo, come dimostra il seguente esempio:

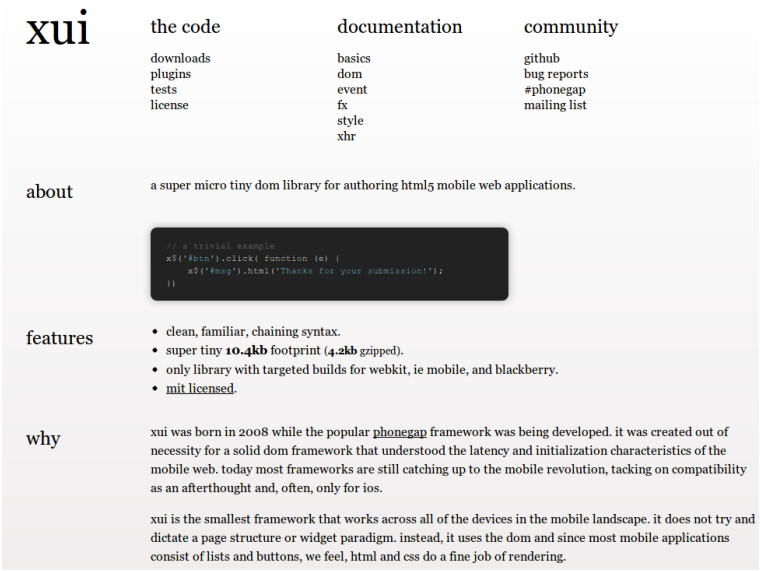
```
//salvataggio di un valore associato ad una chiave
window.localStorage.setItem(key, value);

//recupero dei valori, a partire dalle chiavi
var l = window.localStorage.length;
for (i=0; i < l; i++) {
  storedKey = window.localStorage.key(i);
  storedValue = window.localStorage.getItem(storedKey);
}
```

Local Storage espone una proprietà length e permette di recuperare il valore delle chiavi memorizzate attraverso indici sequenziali. E' possibile quindi scandire i dati salvati con un semplice ciclo.

Local Storage non è l'unica alternativa che gli sviluppatori hanno per ottenere la permanenza dei dati nei browser degli utenti, ma da un punto di vista mobile cross-platform è l'alternativa migliore in quanto supportata da tutti i browser mobile.

3.2 XUI

XUI[8] è una libreria Javascript creata e mantenuta dai fondatori di PhoneGap.



XUI permette di scrivere codice sintatticamente pulito e semplifica l'utilizzo di AJAX per recuperare ed aggiornare in modo asincrono i dati all'interno di una view. XUI realizza un'astrazione che permette allo sviluppatore di ridurre notevolmente la quantità di codice scritto e di renderlo meno application-specific. Seguono alcuni esempi pratici di utilizzo in cui è utile utilizzare le funzioni di questa libreria.

- aggiungere un listener ad un elemento (ad esempio un form)
  - senza XUI

    ```
    document.getElementById("myForm").addEventListener(
            "submit", function (evt) {...});
    ```

  - con XUI

    ```
    x$("#myForm").on("submit", function (evt) {...});
    ```

- inserire codice HTML all'interno di un div
  - senza XUI

    ```
    document.getElementById("colorDescription").
              innerHTML = markup;
    ```

  - con XUI

```
x$(\#colorDescription").inner(markup);
```

- inserire regole di stile per un elemento
  - con XUI

```
x$('a').css({
  'font-weight': 'bold','color': 'white'
});
```

- chiamate AJAX
  - senza XUI

```
var req = new XMLHttpRequest();
req.onreadystatechange = function () {
  if (this.readyState == 4) {
    if (this.status == 200 || this.status == 0){
      ...success code...
    }
  }
}
req.open(\GET", \http://somerequest.com/req?...", true);
req.send();
```

  - con XUI

```
x$().xhr("http://somerequest.com/req?...", function(){
   ...success code... });
```

- tante altre funzioni, ampiamente documentate online nel sito di riferimento.

Negli esempi precedenti emerge con chiarezza come grazie all'utilizzo di questa libreria il codice che lo sviluppatore deve scrivere per effettuare certe operazioni sia quasi dimezzato.

3.3 CSS3

Analogamente a quanto detto per HTML5, le specifiche di CSS3 sono ancora in fase di sviluppo. Nonostante questo, molte feature sono comunque già

state implementate e funzionano discretamente bene anche su applicazioni mobile.

Due feature molto utili che aiutano ad avvicinare il comportamento delle applicazioni sviluppate con framework che utilizzano tecnologie web sono la possibilità di realizzare transizioni tra view e la possibilità di creare semplici animazioni.

3.3.1 Transitions

Le transitions permettono di rendere più gradevole il passaggio tra una view e quella successiva. Generalmente consistono di un effetto di traslazione con cui una view "esce" dallo schermo e una nuova view vi entra. Tali effetti sono usati nativamente da tutte le applicazioni mobile, e permettono di prevenire l'effetto "schermo bianco", che si avrebbe nel momento in cui una pagina viene tolta dalla memoria ed una nuova viene caricata.

Il seguente esempio mostra come creare una view modale. Le view modali sono molto utilizzate in iOS, e si presentano all'utente con una animazione che fa apparire la view dal basso verso l'alto, fino a sovrapporsi completamente a quella precedente.

Per prima cosa si deve assegnare alla view una posizione che sia "fuori" dallo schermo (ad esempio 600 px a partire dall'alto, ma tale valore può variare nel caso dei tablet..).

```
#modal {
  -webkit-transition-duration: 800ms;
  -webkit-transform: translate3d(0,600px,0);
  position: absolute; left: 0; top: 0;
}
```

Poi, tramite codice Javascript, a seguito degli eventi opportuni, si fa partire la transition invocando:

```
function showModal() {
  x$("#modal").css({
    "-webkit-transform":"translate3d(0,0,0)",
    "-webkit-transition-timing-function":"ease-in"
  });
}
```

```
function hideModal() {
  x$("#modal").css({
    "-webkit-transform":"translate3d(0,600px,0)",
    "-webkit-transition-timing-function":"ease-out"
  });
}
```

Tali funzioni non fanno altro che far partire la transition, che consisterà in questo caso nella traslazione verticale delle view modale. Un'altra animazione classica è quella che consiste nella traslazione right-to-left, molto utilizzata in applicazioni navigation-based.

La timing function permette di impostare l'accelerazione con cui deve avvenire la transizione. Tale accelerazione può essere scelta da un insieme di accelerazioni predefinite o può essere implementata manualmente dallo sviluppatore.

3.3.2 Animations

La sintassi per l'implementazione di una animazione è ovviamente più complicata rispetto a quella di una transizione (una transizione non è altro che una semplice animazione).

Un esempio sul come implementare una animazione è il seguente, in cui viene definita una animazione che da un effetto "martellante" all'elemento su cui viene applicata.

```
@-webkit-keyframes throbbing {
  0% {
    background-color: black;
    color: white;
    -webkit-transform: scale(1.0) translate(0,0);
  } 20% {
    -webkit-transform: scale(1.3) translate(40px,0);
  } 50% {
    background-color: white;
    color: black;
    -webkit-transform: scale(1.2) translate(40px,-40px);
  } 80% {
```

```
    -webkit-transform: scale(1.3) translate(0,-40px);
  } 100% {
    background-color: black;
    color: white;
    -webkit-transform: scale(1.0) translate(0,0);
  }
}

h1#pageTitle {
  ...
  -webkit-animation-name: throbbing;
  -webkit-animation-duration: 6s;
  -webkit-animation-direction: alternate;
  -webkit-animation-timing-function: ease-in-out;
  -webkit-animation-iteration-count: infinite;
}
```

Per prima cosa viene definita l'animazione, specificando come devono cambiare le proprietà dell'elemento mentre l'animazione viene eseguita. In seguito l'animazione viene applicata ad un elemento, specificandone la durata, una funzione di timing, e altri parametri.

La definizione di una animazione a prima vista può risultare complessa, ma in realtà è piuttosto semplice, e può essere schematizzata nel seguente modo:

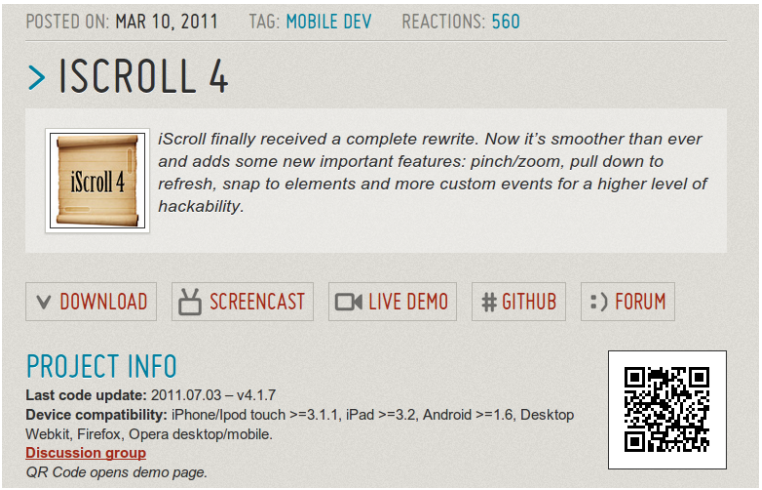
```
@-webkit-keyframes animation-name {
  0% {
  /* original state */
  }
  /* at some point in the middle */ {
  /* properties to animate */
  } 100% {
  /* final state */
  }
}
```

3.3.3 Scrolling

I tipi di transition che sono utilizzati più frequentemente nelle applicazioni mobile sono sicuramente quelli che permettono di fare lo scrolling di una view. La maggior parte delle applicazioni mobile è infatti formata principalmente da:

- una o più parti fisse, che possono ad esempio contenere toolbar, tabbar, menu, ...
- una parte in cui l'utente può spostarsi, con del contenuto che viene nascosto e dell'altro che viene mostrato (ad esempio una tabella con delle immagini e/o informazioni).

Le applicazioni mobile native sono pensate proprio per funzionare in questo modo, dando modo all'utente di spostarsi all'interno delle view. Le applicazioni web invece non sono pensate per avere una parte fissa che resti sempre in primo piano. Tale effetto può essere raggiunto appunto tramite le transizioni. Siccome gestire tale effetto tramite regole CSS non risulta molto comodo, sono state create librerie Javascript a tale scopo, che fanno il lavoro sporco per noi. Tra queste, una molto completa è iScroll[9].



Il seguente esempio mostra realizzare una view con una parte fissa ed una "scrollabile". Per prima cosa bisogna inserire il seguente meta-tag che permette di impostare la larghezza del viewport alla larghezza del device e lo zoom iniziale.

```
<meta name="viewport" content="width=device-width,
        initial-scale=1.0"/>
```

Poi è necessario disabilitare lo scrolling della pagina. In questo modo le parti che devono rimanere fisse non si muoveranno.

```
document.addEventListener('touchmove', function (e) {
  e.preventDefault();
});
```

Il passo successivo è definire un div che contenga il contenuto che vogliamo sia "scrollabile".

```
<div id="wrapper">
  <ul id="list"></ul>
</div>
#wrapper {
  position: relative;
  z-index: 1;
  width: auto;
  height: 150px;
  overflow: scroll;
}
```

Il passo finale consiste nell'inizializzare iScroll, inserendo all'interno del DOMContentLoaded event handler:

```
var scroller = new iScoll(‘list’);
```

In questo modo è possibile ottenere il comportamente desiderato, del tutto analogo a quello delle applicazioni native, anche su applicazioni sviluppate tramite tecnologie web.

3.4 Features Detection

Il Feature Detection è un approccio fondamentale da adottare quando si programma per un insieme eterogeneo di dispositivi come è quello formato dai dispositivi mobile. Tale approccio consiste nel testare l'esistenza di un

oggetto o di una proprietà, ed in base al risultato del test eseguire il codice opportuno.

Ad esempio è possibile testare se le nuove funzionalità HTML5 sono presenti nell'environment corrente, come ad esempio Local Storage. Per testare invece la presenza di una proprietà CSS3, bisogna verificare che sia presente l'attributo nell'oggetto DOM.

Il Feature Detection verrà ampiamente usato in seguito, per verificare ad esempio se sul device su cui sta girando una applicazione supporta o meno l'accesso ad un sensore.

3.5 Media queries

Le Media Queries sono un altro importante strumento che permette di caricare diversi fogli di stile in base al media che accede alla pagina HTML. Siccome i device hanno schermi e risoluzioni anche molto diversi fra loro, è possibile predisporre la visualizzazione degli elementi della pagina in modo diverso a seconda di opportuni parametri. E' evidente che risulta opportuno abbandonare le dimensioni assolute dei componenti che andranno a formare la pagina. Ad esempio, per fare in modo che un div occupi tutto lo schermo:

```
.screen {
  width: 100%;
  height: 100%;
  ...
}
```

Per impostare uno stile personalizzato, ad esempio nel caso in cui il dispositivo sia un iPhone, è sufficiente verificare tramite una media query che la risoluzione sia 320x480.

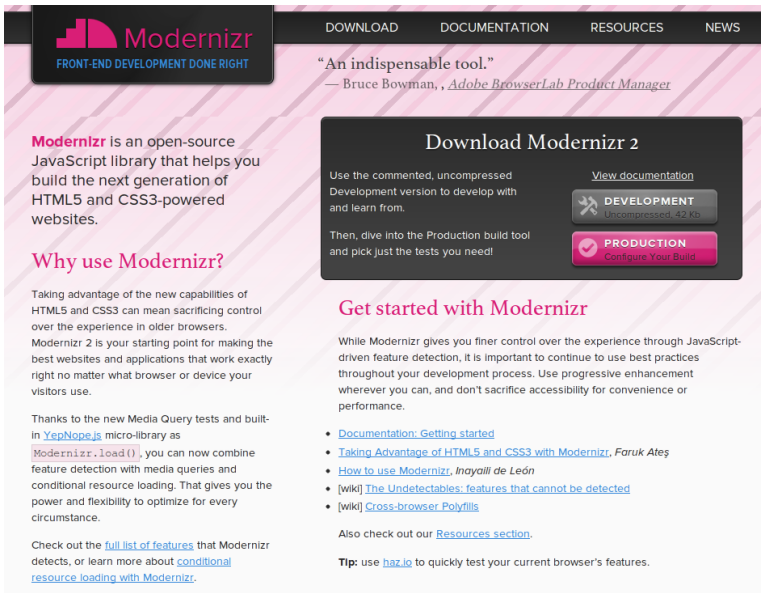
```
@media all and (max-device-width:320px)
          and (max-device-height:480px){
  .screen {
    width: 320px;
    height: 460px;
  }
  ...
}
```

E' possibile creare media queries utilizzando anche altre proprietà, tra cui max-device-width, min-device-width, max-device-height, o min-device-height.

Le media queries sono uno strumento molto potente, utilizzato soprattutto ultimamente dai web developer, per realizzare siti web ottimizzati per i dispositivi mobile, e permettono ad esempio di nascondere/mostrare/spostare contenuti in base al dispositivo/media che accede alla pagina.

Le media queries, combinate con l'esecuzione opzionale di codice Javascript, permettono di realizzare siti web (e quindi anche applicazione mobile) "che si adattano" al contesto in cui vengono eseguiti. Tali tecniche sono dette di Responsive Design e Progressive Enhancements e rappresentano sicuramente il futuro per quel che riguarda il mobile web developing.

L'unico problema che si può riscontrare nell'utilizzo delle Media Queries (e più in generale delle nuove funzionalità HTML5 e CSS3) è che potrebbero non essere supportate completamente da tutti i browser mobile. Anche in questo caso risulta utile il Feature Detection. In particolare, può essere utile utilizzare librerie che permetto agilmente di verificare la presenza o meno di particolari feature. Una tra le più utilizzate è modernizr[10].



3.6 View templating

Il View Templating è un approccio che permette di separare il contenuto di una view da come questa verrà effettivamente mostrata. Il templating dovrà quindi essere usato ogni qualvolta si voglia mostrare qualche informazione, specialmente se ottenuta dinamicamente, all'utente. Fra i tanti framework presenti in rete, una valida scelta è Mustache[11].

Logic-less templates.

Available in Ruby, JavaScript, Python, Erlang, PHP, Perl, Objective-C, Java, .NET, Android, C++, Go, Lua, ooc, ActionScript, ColdFusion, Scala, Clojure, Fantom, CoffeeScript, D, and for node.js.

Works great with TextMate, Vim, Emacs, and Coda.

The Manual: mustache(5) and mustache(1)

Demo

La prima cosa da fare per predisporre il templating sarà definire un template (uno "stampino"), che verrà poi utilizzato per mostrare all'utente dei dati. Tale operazione viene fatto creando un file .mustache con all'interno, ad esempio, il seguente codice:

```
<h2>{{ colorTitle }}</h2>
<p>{{ colorName }} is my favorite color.</p>
```

Questo template sarà utilizzato dal codice Javascript nel momento in cui si vorrà mostrare qualcosa all'utente. Un esempio banale di utilizzo è il seguente:

- Prima si recuperano i dati che si vogliono mostrare e si mettono all'interno di un'oggetto Javascript:

```
var aColorDetail = {
  //definisco i valori da mostrare (fittizzi)
  colorTitle:"Favourite Color",
  colorName:"Red"
};
```

- Poi si invoca una funzione che faccio il "binding" fra il template e il contenuto, come ad esempio la seguente:

```
renderOurTemplate(aColorDetail, function (markup) {
  //inserisco il risultato nel codice html..
  document.getElementById("colorDescription").
        innerHTML = markup;
});
```

che potrebbe essere definita nel seguente modo:

```
var storedTemplate = null;
function renderOurTemplate(view, callback) {
  function doRender(template, view) {
    console.log(\rendering now")
    callback(Mustache.to_html(template, view))
  }
  if (storedTemplate) {
    console.log(\template is stored")
    doRender(storedTemplate, view);
  } else {
    console.log(\template isn't stored");
    var req = new XMLHttpRequest();
    req.onreadystatechange = function () {
      if (this.readyState == 4) {
        if (this.status == 200 ||
                    this.status == 0){
          storedTemplate = this.responseText;
          doRender(storedTemplate, view);
        } else {
          console.log(\something went wrong");
```

```
        }
      }
    }
    req.open(\GET", \color.mustache", true);
    req.send();
  }
}
```

Il View Templating consiste quindi nel prendere un frammento di codice HTML con dei placeholder e rimpiazzare questi con dei contenuti stabiliti a run-time. In questo modo, una volta che si sono creati tutti i template che servono, per mostrare i dati in output, in una view, basta semplicemente invocare una funzione.

I vantaggi di questo approccio sono notevoli, tra cui quello di aver separato la logica dei dati da quella della visualizzazione. Inoltre sarà molto semplice andare a modificare il modo in cui i dati vengono presentati all'utente. Basterà infatti andare a cambiare il codice HTML del file template opportuno.

Capitolo 4

Funzionalità del framework

Dopo aver introdotto e velocemente illustrato le moderne tecniche utilizzate per sviluppare mobile web application, in questo capitolo si procede con la descrizione delle API che il framework mette a disposizione[16], mostrando come sia semplice ed intuitivo il loro utilizzo. Tali API hanno il ruolo fondamentale di wrappare l'accesso alle funzionalità native della piattaforma target, permettendo allo sviluppatore di utilizzare lo stesso codice per tutte le piattaforme.

Prima di procedere con l'elencazione e la descrizione di tutte le funzionalità viene introdotto il meccanismo fondamentale utilizzato dal framework, che utilizza un approccio event-driven.

4.1 Eventi

Nello sviluppo di applicazioni web che utilizzano Javascript è possibile far partire l'esecuzione di determinate operazioni in seguito a ben determinati eventi DOM. Molti elementi presenti nella pagina possono generare eventi, come ad esempio il click su un bottone, il caricamento di una pagina, ...

Il meccanismo utilizzato per associare ad un evento un'azione consiste quindi nel registrare un ascoltatore per quell'evento specifico. Tale ascoltatore, una volta che intercetterà l'evento, dovrà eseguire il codice opportuno. Questo pattern è detto event-listener, ed è molto comune nei linguaggi ad oggetti. Il seguente esempio mostra come lanciare in esecuzione del codice dopo che una pagina web è stata completamente caricata dal browser (l'evento da "ascoltare" è DOMContentLoaded).

```
document.addEventListener("DOMContentLoaded", function () {
    alert("Pagina caricata.");
  }, false);
```

Tale approccio offre diversi vantaggi, tra cui:

- supporto a listener multipli per un evento relativo ad un elemento;
- supporto ad eventi personalizzati;
- semplicità di registrazione/deregistrazione di un ascoltatore.

Per quel che riguarda la programmazione con PhoneGap, il meccanismo è del tutto analogo. PhoneGap mette a disposizione degli sviluppatori un certo numero di eventi (che non fanno parte di quelli DOM, e che quindi non sono utilizzabili dai browser). Nel seguito viene fatta una panoramica tra gli eventi più importanti che bisognerebbe gestire per fare che in modo che il lifecycle delle applicazioni PhoneGap sia paragonabile a quello delle applicazioni native.

Occorre tenere in considerazione il fatto che gli eventi non sono disponibili in maniera omogenea per tutte le piattaforme e bisogna quindi consultare la documentazione online prima di effettuare il building di una applicazione per assicurarsi che gli eventi utilizzati siano effettivamente presenti ed utilizzabili.

4.1.1 deviceready

Questo evento è sicuramente l'evento più importante fra gli eventi custom di PhoneGap. Viene lanciato dall'oggetto document dopo che le librerie di PhoneGap sono state completamente caricate. Dopo che tale evento viene lanciato è quindi possibile utilizzare le funzioni di PhoneGap. devideready viene sempre lanciato dopo DOMContentLoaded. Se si ha del codice che manipola semplicemente la struttura DOM della pagina, questo può essere inserito nel handler dell'evento DOMContentLoaded, mentre se tale codice effettua anche delle chiamate a funzioni PhoneGap, è necessario inserirlo nel handler dell'evento deviceready. Segue un semplice esempio che illustra quanto spiegato in precedenza.

```
<!DOCTYPE html>
<html>
  <head>
    <title>PhoneGap Device Ready Example</title>
    <script type="text/javascript" src="phonegap.js"></script>
    <script type="text/javascript">

    // the document has loaded but phonegap.js has not.
    // When PhoneGap is loaded and talking
    // with the native device,
    // it will call the event ‘deviceready‘.
    function onLoad() {
        document.addEventListener("deviceready",
                            onDeviceReady, false);
    }

    // PhoneGap is loaded and it is now safe
    // to make calls PhoneGap methods
    function onDeviceReady() {
        // Now safe to use the PhoneGap API
        // Here we can register Listeners for all the events!
    }
    </script>
  </head>
  <body onload="onLoad()"> ... </body>
</html>
```

Dopo che viene lanciato l'evento deviceready, è opportuno registrare tutti i listener interessati ad ascoltare gli altri eventi custom di PhoneGap. Nel seguito vengono descritti gli altri eventi che PhoneGap mette a disposizione.

4.1.2 backbutton, menubutton, searchbutton

Nei device dotati di tasti fisici, come ad esempio i telefoni Android e BlackBerry, alla pressione di tali tasti corrisponde il lancio di eventi. PhoneGap permette di intercettare i seguenti eventi:

- backbutton, lanciato alla pressione del tasto indietro;

- menubutton, lanciato alla pressione del tasto menu;
- searchbutton, lanciato alla pressione del tasto cerca.

Segue un semplicissimo esempio relativo al primo evento.

```
//da mettere all'interno di onDeviceReady
document.addEventListener("backbutton", onBackKeyDown, false);

function onBackKeyDown() {
    // Handle the back button
}
```

In questo modo lo sviluppatore può sovrascrivere ed impostare il comportamento di tali tasti fisici come ritiene opportuno e logico, ed assegnare a tali tasti un comportamento analogo a quello che avrebbero in applicazioni native.

4.1.3 pause, resume

Gli eventi pause e resume vengono lanciati rispettivamente quando l'applicazione viene messa in background/tolta dal background. Come nei casi precedenti, il loro utilizzo è molto semplice.

```
document.addEventListener("pause", onPause, false);

function onPause() {
    // Handle the pause event
}
```

4.1.4 online, offline

Gli eventi online e offline vengono lanciati rispettivamente quando il device si collega/scollega ad internet.

```
document.addEventListener("online", onOnline, false);

function onOnline() {
    // Handle the online event
}
```

4.1.5 altri eventi

Altri eventi meno importanti ma comunque utili sono i seguenti:

- batterycritical, lanciato quando il livello di carica della batteria del device raggiunge la soglia critica (tale valore dipende dal device);
- batterylow, simile al precedente, lanciato quando il livello di carica della batteria è basso;
- batterystatus, lanciato quando il livello di carica della batteria cambia;
- startcallbutton/endcallbutton, lanciati rispettivamente quando viene premuto il tasto fisico verde/rosso per iniziare/terminare una chiamata;
- volumedownbutton e volumeupbutton, lanciati rispettivamente quando vengono premuti i tasti fisici per alzare/abbassare il volume.

4.2 Funzionalità native

La vera forza di PhoneGap consiste nel fornire l'accesso a funzionalità native, presentandole all'utente nello stesso modo in cui verrebbero presentate dalle applicazioni native. Le funzionalità native più importanti sono quelle riguardanti il recupero e l'utilizzo dei dati provenienti dai sensori.

Un sensore è un elemento hardware che permette di recuperare informazioni riguardanti l'ambiente in cui si trova il device, e può essere o meno presente in uno specifico device. I sensori possono essere interrogati in due modi:

- in maniera asincrona, invocando:

  ```
  navigator.sensor.getCurrentVariable(successCallback,
            errorCallback, options)
  ```

  La funzione interroga il sensore ed una volta che questo le restituisce i dati, questi vengono passati come primo parametro alla funzione di callback.

- periodicamente, invocando:

```
navigator.sensor.watchVariable(successCallback,
        errorCallback, options)
```

La funzione chiama periodicamente quella precedente (una sorta di polling) e passa il risultato al callback ogni volta. L'intervallo che specifica ogni quanto avviene l'interrogazione al sensore deve essere settato in options. La funzione ritorna una varibale watchId, che può essere usata ad esempio per terminare il polling, nel seguente modo: navigator.sensor.clearWatch(watchId)

Recuperare dati da un sensore risulta un'operazione ben standardizzata e consistente per ogni diverso sensore. L'oggetto di risposta che sarà passato al callback sarà ovviamente formato da campi che dipendono dal sensore interrogato.

Bisogna sempre tenere in considerazione che, come mostrato in precedenza, non tutte le piattaforme e device hanno a disposizione gli stessi sensori. Occorre quindi prestare molta attenzione nel momento in cui si deve eseguire il building di una applicazione su piattaforme diverse.

Quella che segue è una semplice panoramica che vuole rendere l'idea delle potenzialità che il framework mette a disposizione degli sviluppatori. Per forza di cose, tale panoramica non potrà coprire tutti i dettagli delle librerie di PhoneGap. Per tali dettagli si rimanda alla documentazione ufficiale.

4.2.1 Accelerometer

Un accelerometro è un sensore che misura la posizione in cui si trova il device in uno spazio tridimensionale. Tale sensore permette di monitorare le coordinate x, y e z della posizione nel tempo. Seguono due esempi, corrispondenti ai due possibili modi di interrogare il sensore.

1. getCurrentVariable

```
<!DOCTYPE html>
<html>
  <head>
    <title>Acceleration Example</title>
    <script type="text/javascript" src="phonegap.js">
    </script>
```

```
        <script type="text/javascript" charset="utf-8">

        // Wait for PhoneGap to load
        document.addEventListener("deviceready",
                    onDeviceReady, false);

        function onDeviceReady() {
           // PhoneGap is ready
           navigator.accelerometer.getCurrentAcceleration
                (onSuccess, onError);
        }

        // onSuccess: Get a snapshot of the current acc
        function onSuccess(acceleration) {
            alert('Acc X: ' + acceleration.x + '\n' +
                  'Acc Y: ' + acceleration.y + '\n' +
                  'Acc Z: ' + acceleration.z + '\n' +
                  'Timestamp: ' + acceleration.timestamp);
        }

        // onError: Failed to get the acceleration
        function onError() {
            alert('onError!');
        }

        </script>
      </head>
      <body>
        <h1>Example</h1>
        <p>getCurrentAcceleration</p>
      </body>
    </html>
```

2. watchVariable

```
    <!DOCTYPE html>
    <html>
```

```
<head>
  <title>Acceleration Example</title>
  <script type="text/javascript" src="phonegap.js">
  </script>
  <script type="text/javascript">
  // The watch id references the ‘watchAcceleration‘
  var watchID = null;
  // Wait for PhoneGap to load
  document.addEventListener("deviceready",
                   onDeviceReady, false);

  function onDeviceReady() {
      startWatch();
  }

  // Start watching the acceleration
  function startWatch() {
      // Update acceleration every 3 seconds
      var options = { frequency: 3000 };
      watchID = navigator.accelerometer.
          watchAcceleration(
            onSuccess, onError, options);
  }

  // Stop watching the acceleration
  function stopWatch() {
      if (watchID) {
          navigator.accelerometer.clearWatch(watchID);
          watchID = null;
      }
  }

  function onSuccess(acc) {
      var element = document.
          getElementById('accelerometer');
     element.innerHTML='Acc X: '+acc.x+'<br />'+
                       'Acc Y: '+acc.y+'<br />'+
```

```
                        'Acc Z: '+acc.z+'<br />'+
                        'Timestamp: '+acc.timestamp;
    }

    function onError() {
        alert('onError!');
    }
    </script>
  </head>
  <body>
    <div id="accelerometer">Waiting for accelerometer</div>
  </body>
</html>
```

In entrambi i due metodi, è possibile iniziare a monitorare l'accelerometro solo dopo che è stato lanciato l'evento deviceready. Negli esempi seguenti questo verrà dato per scontato. L'oggetto acceleration passato al callback nel caso in cui la lettura sia andata a buon fine è formato dalle seguenti proprietà:

- x: amount of motion on the x-axis. Range [0, 1] (Numero)
- y: amount of motion on the y-axis. Range [0, 1] (Numero)
- z: amount of motion on the z-axis. Range [0, 1] (Numero)
- timestamp: creation timestamp in milliseconds. (DOMTimeStamp)

Utilizzando i valori recuperati dall'accelerometro è possibile ricostruire i movimenti che il device ha compiuto e sfruttare tali informazioni per offrire delle feature particolari, come ad esempio riconoscere se l'utente ha fatto uno "shake", ...

4.2.2 Geolocation

La geolocalizzazione fornisce la longitudine e la latitudine a cui si trova il device. Tali valori possono essere recuperati attraverso il sensore GPS installato nel device, o attraverso l'indirizzo IP, ... Analogamente a prima, seguono due esempi di utilizzo.

1. getCurrentVariable

```
// onSuccess Callback
//   This method accepts a ‘Position‘ object,
//   which contains the current GPS coordinates
var onSuccess = function(pos) {
    alert('Latitude: ' + pos.coords.latitude  + '\n' +
          'Longitude: '+ pos.coords.longitude + '\n' +
          'Altitude: ' + pos.coords.altitude  + '\n' +
          'Accuracy: ' + pos.coords.accuracy  + '\n' +
          'AltAccuracy:'+pos.coords.altitudeAccuracy+'\n'+
          'Heading: '  + pos.coords.heading   + '\n' +
          'Speed: '    + pos.coords.speed     + '\n' +
          'Timestamp: '+ new Date(pos.timestamp)+'\n');
};

// onError Callback receives a PositionError object
function onError(error) {
    alert('code: '    + error.code    + '\n' +
          'message: ' + error.message + '\n');
}

navigator.geolocation.getCurrentPosition(
        onSuccess, onError);
```

2. watchVariable

```
// onSuccess Callback
//   This method accepts a ‘Position‘ object,
//    which contains the current GPS coordinates
function onSuccess(position) {
    var element = document.getElementById('geolocation');
    element.innerHTML =
        'Latitude: ' + position.coords.latitude  + '<br />' +
        'Longitude: '+ position.coords.longitude + '<br />' +
        '<hr />'     + element.innerHTML;
}
```

```
// onError Callback receives a PositionError object
function onError(error) {
    alert('code: '    + error.code    + '\n' +
          'message: ' + error.message + '\n');
}

// Options: retrieve the location every 3 seconds
var watchID = navigator.geolocation.watchPosition(
                    onSuccess, onError,
                    { frequency: 3000 });

...
navigator.geolocation.clearWatch(watchID);
```

L'oggetto position passato al callback è formato dalle seguenti proprietà:

- coords: A set of geographic coordinates. (Coordinates)
- timestamp: Creation timestamp for coords in milliseconds. (DOMTimeStamp)

L'oggetto Coordinates è a sua volta costituito dalle seguenti proprietà:

- latitude: Latitude in decimal degrees. (Number)
- longitude: Longitude in decimal degrees. (Number)
- altitude: Height of the position in meters above the ellipsoid. (Number)
- accuracy: Accuracy level of the latitude and longitude coordinates in meters. (Number)
- altitudeAccuracy: Accuracy level of the altitude coordinate in meters. (Number)
- heading: Direction of travel, specified in degrees counting clockwise relative to the true north. (Number)

- speed: Current ground speed of the device, specified in meters per second. (Number)

Utilizzando la posizione corrente del device è possibile ad esempio mostrare all'utente la sua posizione precisa all'interno di una mappa, regolare le preferenze di una applicazione sulla base del paese in cui si trova, ...

4.2.3 Camera

L'oggetto camera fornisce l'accesso all'applicazione nativa della fotocamera. Tale oggetto, diversamente da tutti gli altri, possiede solo un metodo:

```
navigator.camera.getPicture( cameraSuccess, cameraError,
        [ cameraOptions ] );
```

Tale metodo permette di scattare una foto o di recuperarla dagli album presenti nel device. L'immagine recuperata sarà nel formato String base64 o corrisponderà all'URI dell'immagine all'interno del dispositivo. Segue un esempio di codice completo, che illustra tutte le possibili opportunità che questo oggetto mette a disposizione.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Capture Photo</title>
    <script type="text/javascript" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

    var pictureSource;   // picture source
    var destinationType; // sets the format of returned value

    // Wait for PhoneGap to connect with the device
    document.addEventListener("deviceready",
            onDeviceReady,false);

    // PhoneGap is ready to be used!
    function onDeviceReady() {
        pictureSource=navigator.camera.PictureSourceType;
```

```
    destinationType=navigator.camera.DestinationType;
}

// Called when a photo is successfully retrieved
function onPhotoDataSuccess(imageData) {
  // Uncomment to view the base64 encoded image data
  // console.log(imageData);

  // Get image handle
  var smallImage = document.getElementById('smallImage');

  // Unhide image elements
  smallImage.style.display = 'block';

  // Show the captured photo
  // The inline CSS rules are used to resize the image
  smallImage.src = "" + imageData;
}

// Called when a photo is successfully retrieved
function onPhotoURISuccess(imageURI) {
  // Uncomment to view the image file URI
  // console.log(imageURI);

  // Get image handle
  var largeImage = document.getElementById('largeImage');

  // Unhide image elements
  largeImage.style.display = 'block';

  // Show the captured photo
  // The inline CSS rules are used to resize the image
  largeImage.src = imageURI;
}

// A button will call this function
function capturePhoto() {
```

```
    // Take picture using device camera and retrieve
    // image as base64-encoded string
    navigator.camera.getPicture(onPhotoDataSuccess,
                      onFail, { quality: 50 });
  }

  // A button will call this function
  function capturePhotoEdit() {
    // Take picture using device camera, allow edit,
    // and retrieve image as base64-encoded string
    navigator.camera.getPicture(onPhotoDataSuccess,
            onFail, { quality: 20, allowEdit: true });
  }

  // A button will call this function
  function getPhoto(source) {
    // Retrieve image file location from specified source
    navigator.camera.getPicture(onPhotoURISuccess,
                            onFail, { quality: 50,
      destinationType: destinationType.FILE_URI,
      sourceType: source });
  }

  // Called if something bad happens.
  function onFail(message) {
    alert('Failed because: ' + message);
  }

  </script>
</head>
<body>
  <button onclick="capturePhoto();">
      Capture Photo</button> <br>
  <button onclick="capturePhotoEdit();">
      Capture Editable Photo</button> <br>
  <button onclick="getPhoto(pictureSource.PHOTOLIBRARY);">
      From Photo Library</button><br>
```

```
    <button onclick="getPhoto(pictureSource.SAVEDPHOTOALBUM);">
        From Photo Album</button><br>
    <img style="display:none;width:60px;height:60px;"
         id="smallImage" src="" />
    <img style="display:none;" id="largeImage" src="" />
  </body>
</html>
```

Per scattare una foto è necessario settare la proprietà sourceType dell'oggetto camera con la costante Camera.PictureSourceType.CAMERA, mentre invece se si vuole recuperare una foto dal photoalbum è necessario settare tale proprietà con Camera.PictureSourceType.PHOTOLIBRARY o Camera.PictureSourceType.SAVEDPHOTOALBUM. Il valore di ritorno sarà passato come parametro alla funzione di callback come stringa base64 o come stringa che rappresenta l'URI della foto nel filesystem del device, in base al valore settato nei parametri opzionali. Una volta che si è recuperata la foto, è possibile mostrarla in una view, salvarla, mandarla ad un server, ...

4.2.4 Contacts

PhoneGap permette l'accesso in lettura e scrittura al database interno del telefono che contiene la rubrica dell'utente. I metodi che vengono messi a disposizione per questo scopo sono due:

- contacts.create(properties); Questo metodo crea un oggetto contact. Una volta creato tale oggetto è possibile assegnarvi delle proprietà. Dopo aver settato tutte le proprietà, per salvare il contatto creato nella rubrica del telefono è sufficiente invocare il metodo save.

```
var myContact = navigator.contacts.create(
     {"displayName": "Test User"});
myContact.gender = "male";
myContact.save(onsuccess,onfail);
console.log("The contact, " +
     myContact.displayName + ", is of the " +
     myContact.gender + "gender");
```

- contacts.find(contactFields, contactSuccess, contactError, contactFindOptions); Questa metodo è asincrono e restituisce un array di oggetti Contact, che viene passato come parametro alla funzione di callback. Nel primo parametro (contactFields) deve essere passato un'array contenente il nome dei campi che si vogliono ottenere (ad es: name, phonenumber, emails). Nell'ultimo parametro (contactFindOption) è possibile specificare delle opzioni che riguardano la ricerca, come ad esempio dei filtri personalizzati di ricerca.

```
function onSuccess(contacts) {
    alert('Found ' + contacts.length + ' contacts.');
};

function onError(contactError) {
    alert('onError!');
};

// find all contacts with 'Bob' in any name field
var options = new ContactFindOptions();
options.filter="Bob";
var fields = ["displayName", "name"];
navigator.contacts.find(fields, onSuccess,
        onError, options);
```

L'oggetto Contact ha numerose proprietà e anche qualche metodo per manipolarne lo stato interno. Una applicazione potrà quindi andare a modificare un contatto nella rubrica. Le proprietà sono:

- id: A globally unique identifier. (DOMString)
- displayName: The name of this Contact, suitable for display to end-users. (DOMString)
- name: An object containing all components of a persons name. (ContactName)
- nickname: A casual name to address the contact by. (DOMString)
- phoneNumbers: An array of all the contact's phone numbers. (ContactField[])

- emails: An array of all the contact's email addresses. (ContactField[])
- addresses: An array of all the contact's addresses. (ContactAddresses[])
- ims: An array of all the contact's IM addresses. (ContactField[])
- organizations: An array of all the contact's organizations. (ContactOrganization[])
- birthday: The birthday of the contact. (Date)
- note: A note about the contact. (DOMString)
- photos: An array of the contact's photos. (ContactField[])
- categories: An array of all the contacts user defined categories. (ContactField[])
- urls: An array of web pages associated to the contact. (ContactField[])

I metodi:

- clone: Returns a new Contact object that is a deep copy of the calling object, with the id property set to null.
- remove: Removes the contact from the device contacts database. An error callback is called with a ContactErrorobject if the removal is unsuccessful.
- save: Saves a new contact to the device contacts database, or updates an existing contact if a contact with the same idalready exists.

4.2.5 Capture

PhoneGap permette di recuperare audio, video e immagini in serie, interfacciandosi con le applicazioni native di sistema, tramite delle interfacce generiche, che formano le API Media Capture.

Le API che permettono di recuperare delle immagini, ad esempio, funzionano in maniera simile alla funzione getPicture precedentemente introdotta. Il punto di forza di queste API è che si presentano in maniera standard, sia

che si tratti di registrare audio, video o di recuperare immagini scattate. Sfortunatamente le API audio e video di HTML5 non sono ancora sviluppate a dovere, percui per eseguire operazioni di manipolazione audio/video occorre utilizzare codice nativo.

Le tre funzioni fondamentali da invocare per utilizzare queste API sono:

- per effettuare registrazioni audio:

```
navigator.device.capture.captureAudio(
    CaptureCB captureSuccess, CaptureErrorCB captureError,
    [CaptureAudioOptions options]
);
```

- per effettuare registrazioni video:

```
navigator.device.capture.captureVideo(
    CaptureCB captureSuccess, CaptureErrorCB captureError,
    [CaptureVideoOptions options]
);
```

- per recuperare immagini:

```
navigator.device.capture.captureImage(
    CaptureCB captureSuccess, CaptureErrorCB captureError,
    [CaptureImageOptions options]
);
```

Queste fuzioni hanno numerose opzioni ed oggetti che permettono di configurarne il comportamente. Per ogni ulteriore dettaglio si rimanda alla documentazione ufficiale.

4.2.6 Notifications

PhoneGap utilizza le librerie native anche per quanto riguarda le notifiche visuali, sonore e tattili. I metodi messi a disposizione sono i seguenti quattro:

- navigator.notification.alert(message, alertCallback, [title], [buttonName]): mostra semplicemente un dialog box. Un semplice esempio di utilizzo è il seguente.

```
function alertDismissed() {
    // do something
}

navigator.notification.alert(
    'You are the winner!',  // message
    alertDismissed,         // callback
    'Game Over',            // title
    'Done'                  // buttonName
);
```

- navigator.notification.confirm(message, confirmCallback, [title], [buttonLabels]): molto simile al precedente, permette l'esecuzione di codice in seguito alla pressione di un tasto di conferma.

```
// process the confirmation dialog result
function onConfirm(button) {
    alert('You selected button ' + button);
}

// Show a custom confirmation dialog
function showConfirm() {
    navigator.notification.confirm(
        'You are the winner!',  // message
        onConfirm,              // callback
        'Game Over',            // title
        'Restart,Exit'          // buttonLabels
    );
}
```

- navigator.notification.beep(times): riproduce dei beep di sistema.

```
// Beep twice!
navigator.notification.beep(2);
```

- navigator.notification.vibrate(milliseconds): fa vibrare il device.

```
// Vibrate for 2.5 seconds
navigator.notification.vibrate(2500);
```

L'utilizzo di queste librerie è molto semplice ed intuitivo.

4.2.7 Compass

L'oggetto compass permette di ottenere la direzione verso cui il device è rivolto. Tale valore è espresso in gradi, e il suo range va da 0 a 359.99. Questo oggetto può essere interrogato nei due modi introdotti in precedenza. Seguono due semplici esempi.

1. getCurrentVariable

```
function onSuccess(heading) {
    alert('Heading: ' + heading.magneticHeading);
};

function onError(error) {
    alert('CompassError: ' error.code);
};

navigator.compass.getCurrentHeading(onSuccess, onError);
```

2. watchVariable

```
function onSuccess(heading) {
    var element = document.getElementById('heading');
    element.innerHTML = 'Heading: ' +
        heading.magneticHeading;
};

function onError(compassError) {
        alert('Compass error: ' + compassError.code);
};
```

```
// Update every 3 seconds
var options = { frequency: 3000 };
var watchID = navigator.compass.watchHeading(onSuccess,
              onError, options);

// ... later on ...
navigator.compass.clearWatch(watchID);
```

Anche in questo caso, la standardizzazione delle API di PhoneGap rende semplicissimo il recupero dei valori dal device.

4.2.8 Altre API

Oltre alle API precedentemente descritte, ve ne sono anche altre di cui si da una breve descrizione senza entrare troppo nei dettagli.

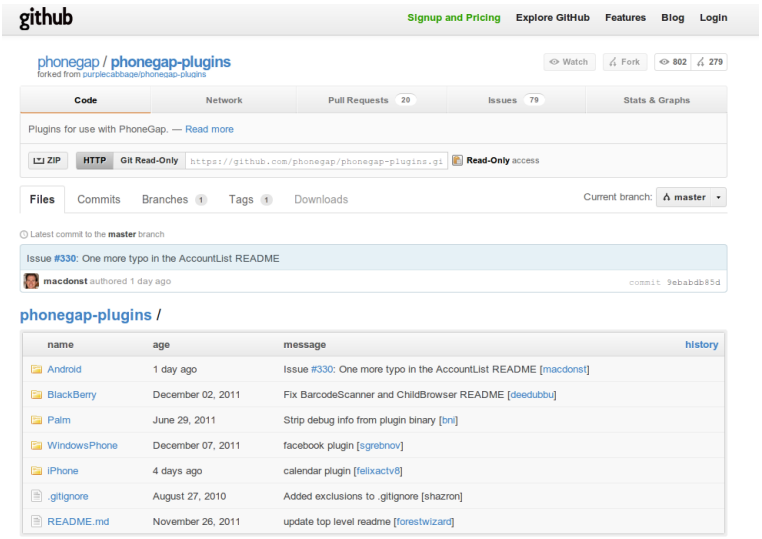
- Device: queste API permettono di ottenere informazioni riguardanti il software e l'hardware che è utilizzato dal device.
- Connection: queste API permettono di ottenere informazioni riguardante il tipo di rete a cui il device è collegato (wifi, 2g, 3g, ...).
- Media: queste API permettono di registrare e riprodurre file audio.
- File Queste API permettono di leggere, scrivere e navigare il filesystem del devices.
- Storage: queste API permettono di creare ed interagire un database interno all'applicazione, basato sulla specifica W3C Web SQL Database.

Per tali librerie si rimanda alla documentazione ufficiale.

4.3 Plugins

Nei paragrafi precedenti sono state illustrate le principali API che PhoneGap mette a disposizione degli sviluppatori. Tali API non sono altro che "connettori", che permettono l'accesso a funzionalità native tramite precise interfacce Javascript. Le API di PhoneGap non possono però coprire interamente tutte le funzionalità native presenti in tutte le piattaforme.

Per sopperire a questo limite, PhoneGap mette a disposizione una robusta architettura che permette agli sviluppatori di scrivere plugin personalizzati, che potranno essere integrati nelle applicazioni e portati in tutte le piattaforme. Ogni sviluppatore può quindi estendere le funzionalità base di PhoneGap scrivendo plugin e contribuire al framework condividendoli con la comunità.



Ogni plugin deve ovviamente essere sviluppato indipendentemente per ogni diversa piattaforma. Sarà compito dello sviluppatore avere l'accortezza di fornire una interfaccia Javascript standard per tutte le piattaforme.

Capitolo 5

Un caso di studio: Visage

Come caso di studio per testare ed applicare le tecniche precedentemente descritte viene ripreso un progetto sviluppato per un precedente corso universitario, Visage. Tale progetto viene brevemente introdotto nel primo paragrafo. Nel secondo paragrafo viene posta l'attenzione sulla parte rilevante per questa tesi, ovvero su come l'applicazione mobile dovrà comportarsi ed interfacciarsi verso i preesistenti servizi web. All'interno del terzo e del quarto paragrafo vengono mostrati gli screenshots dell'applicazione nativa e di quella sviluppata con PhoneGap. Sempre nel quarto paragrafo, viene presentata una analisi delle funzionalità e ne viene proposta una soluzione implementativa. Nel paragrafo conclusivo si farà il punto della situazione una volta terminata l'implementazione dell'applicazione. In questo capitolo si fornisce una descrizione del problema.

5.1 Descrizione dell'applicazione

La parte server che supporta tale progetto è gia stata implementata, cosi come l'applicazione Android. Nel seguito viene descritto il sistema in generale, tralasciando la parte di implementazione della parte server, non di competenza di questa tesi. Per quel che concerne questa tesi, basti sapere che l'architettura del sistema è una semplice architettura REST. L'applicazione mobile potrà quindi accedere ai servizi del server tramite richieste HTTP opportunamente formattate.

5.1.1 Descrizione generale

Il progetto consiste nella realizzazione di una piattaforma web social in cui ogni utente si registra rendendo pubblici (all'interno della piattaforma) i propri account social (email, twitter, facebook, etc). Ogni utente potrà quindi accedere alle informazioni di una qualsiasi persona tramite il riconoscimento facciale, a patto che essa sia registrata. Il riconoscimento potrà avvenire:

- lato desktop con l'upload di una foto attraverso l'applicazione web
- lato mobile sempre tramite l'upload di una foto e anche attraverso un sistema di realtà aumentata.

Il progetto prevede la realizzazione di:

- una applicazione web, in cui gli utenti potranno registrarsi, modificare i propri dati, cancellare ed aggiungere le proprie foto, caricare foto per riconoscere le persone presenti.
- applicazioni mobile (Android e iOS), in cui gli utenti potranno registrarsi, modificare i propri dati, richiedere il riconoscimento tramite il caricamento di foto o tramite realtà aumentata.

5.1.2 Estensioni

Integrazione di un servizio che tracci la posizione e indichi all'utente quali utenti sono vicini a lui, o che dia comunque la possibilità di limitare la ricerca tra gli utenti che sono effettivamente nei paraggi.

5.1.3 Server

Il server si deve occupare di gestire tutte le richieste che giungono dai client, desktop e mobile, e di interfacciarsi con le API di face.com (servizio che viene attualmente utilizzato per gli algoritmi di riconoscimento facciale). Come precedentemente detto, non verranno descritti i dettagli implementativi di questa parte, ma verrà semplicemente descritta l'interfaccia con cui i client mobile interegiscono con il server.

5.1.4 Client

I vari client, che siano mobile o desktop, possono interrogare il server e richiedere ad esempio la registrazione di un nuovo utente, la modifica dei dati personali di un utente, il riconoscimento di un utente data una foto, la lista degli utenti nelle vicinanze...

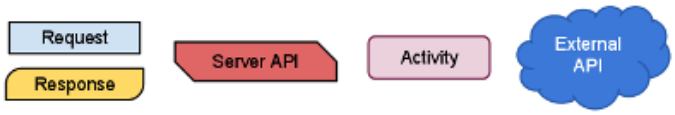
5.1.5 Registrazione

Al momento dell'iscrizione verranno chiesti (tutti gli elementi sono obbligatori):

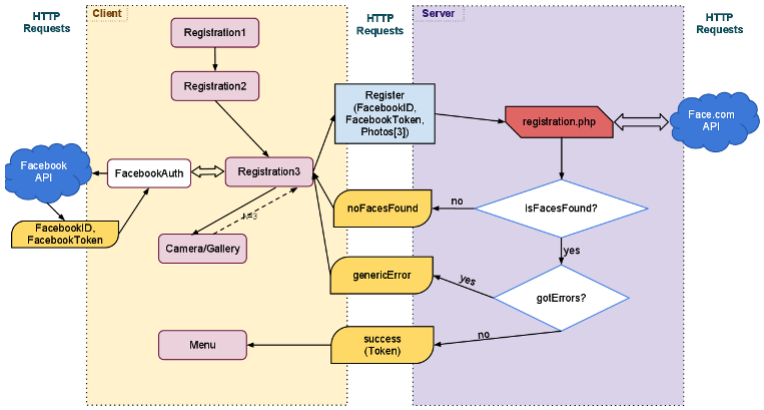
- Nome e Cognome, Mail e Password, Data di nascita
- Almeno 3 foto in cui è presente solo l'utente (una sorta di immagine del profilo). In questo modo il sistema avrà a disposizione un numero sufficiente di foto dell'utente, che saranno utilizzate per riconoscere l'utente stesso.
- Collegamento con account Facebook tramite login per avere conferma dell'identità.

5.2 Descrizione dell'interazione MobileClient - Server

I seguenti diagrammi mostrano come l'applicazione dovrà interagire con il server durante le tre principali operazioni che dovrà compiere, ovvero registrazione, login e riconoscimento. Nei seguenti paragrafi verranno illustrati semplici diagrammi che hanno l'intento di spiegare il comportamento dell'applicazione e la sua interazione con il server. Per tali diagrammi valgono le convenzioni illustrate nella seguente figura.



5.2.1 Registrazione



Il processo di registrazione è relativamente complesso, visti i vincoli imposti in fase di redazione dei requisiti. Per semplificare tale processo si è scelto di suddividerlo in tre sotto-processi:

1. nella prima parte vengono semplicemente richieste all'utente le sue informazioni di base;

2. nella seconda parte viene richiesto di effettuare il login tramite facebook, di modo da confermare la propria identità;

3. nella terza parte viene richiesto di scegliere tre foto tra quelle presenti nella galleria del proprio telefono, o di scattare le foto sul momento tramite la fotocamera.

Il processo di registrazione dovrà contattare le API di Facebook per prelevare alcune informazione relative all'utente e anche il server per effettuare l'upload e il controllo delle foto. In ogni foto dovrà infatti essere presente la faccia dell'utente. Una volta avvenuto tale controllo per tutte le foto, il processo di registrazione potrà proseguire. Una volta terminato il processo di registrazione, l'utente sarà invitato a controllare la propria mail, per attivare il proprio account appena creato.

Segue il formato delle richieste/risposte tra MobileClient e Server durante il processo di registrazione.

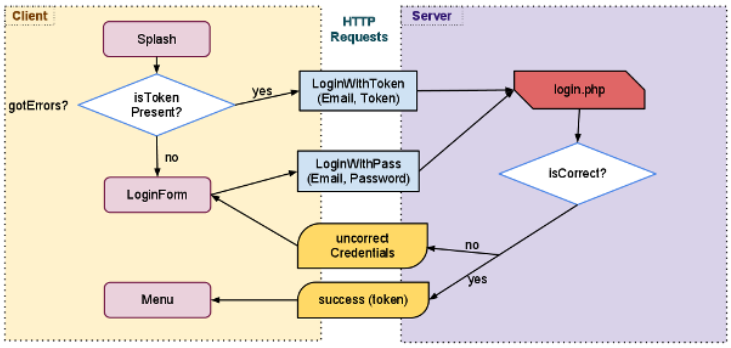
- uploadpic.php

Tipo	Campo	Descrizione
Req		
	fbuid	Facebook ID dell'utente a cui la foto si riferisce
	picindex	indice della foto (es: 1, o 2, o 3)
	image	immagine da controllare
Resp		
	result	risultato della richiesta
	picindex	indice della foto

- registration.php

Tipo	Campo	Descrizione
Req		
	name	nome dell'utente
	surname	cognome dell'utente
	dateBirth	data di nascita dell'utente
	email	email dell'utente
	password	password dell'utente
	facebooktoken	token Facebook dell'utente
	uid	ID Facebook dell'utente
Resp		
	result	risultato della richiesta
	error	descrizione dell'errore

5.2.2 Login



Il login al servizio, per come è stato progettato originariamente il sistema, deve avvenire ogni volta che l'utente apre l'applicazione. Per prima cosa

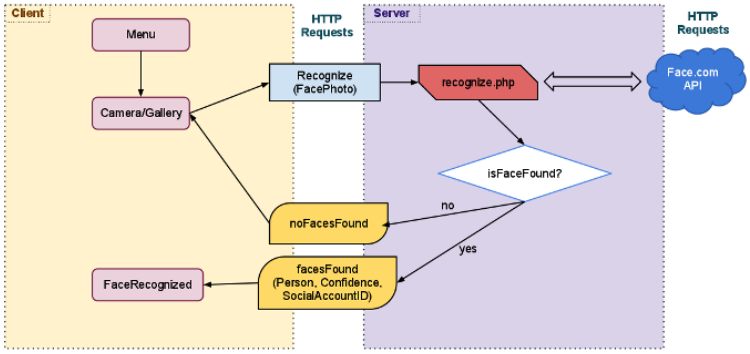
viene verificato se è presente un token dell'applicazione, ed in tal caso viene tentato il login con tale codice. Se il token non è presente all'utente devono essere richieste le credenziali di accesso.

Segue il formato delle richieste/risposte tra MobileClient e Server durante il processo di registrazione.

- login.php

Tipo	Campo	Descrizione
Req		
	email	email dell'utente
	token	token dell'applicazione
	password	password dell'utente
Resp		
	result	risultato della richiesta
	name	nome dell'utente
	surname	cognome dell'utente
	datebirth	data di nascita dell'utente
	photo	URL della foto del profilo dell'utente
	mobiletoken	nuovo token generato per l'utente

5.2.3 Riconoscimento



Anche il processo di riconoscimento è relativamente semplice. Per prima cosa l'applicazione dovrà acquisite una immagine (scattare una foto, oppure prenderla dalla galleria dell'utente, oppure prelevare un frame dalla fotocamera). Il passo successivo sarà l'invio al server dell'immagine e la richiesta di effettuare il riconoscimento. I server elaboreranno la richiesta

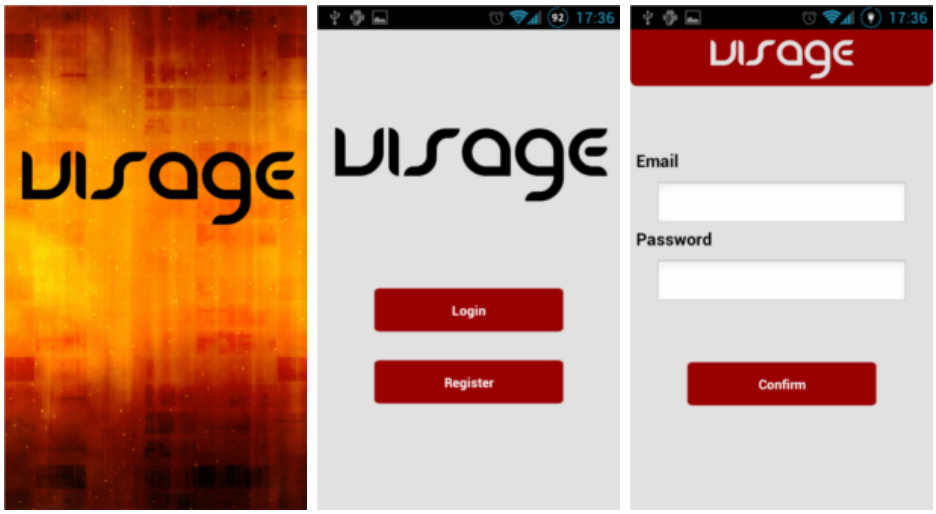
restituendo le informazioni relative all'utente riconosciuto in caso positivo, oppure un messaggio di errore.

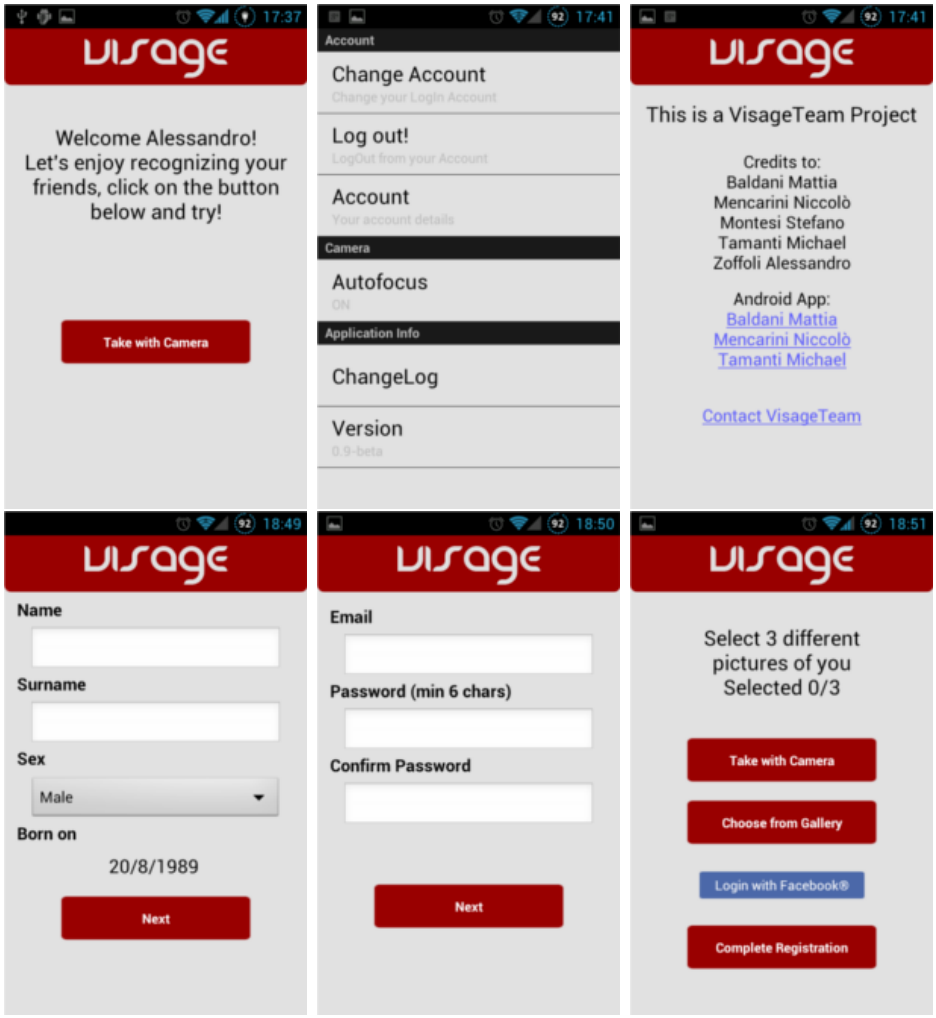
- recognize.php

Tipo	Campo	Descrizione
Req		
	email	email dell'utente
	token	token dell'applicazione
	uploadedfile	immagine su cui effettuare il riconoscimento
Resp		
	result	risultato della richiesta
	name	nome dell'utente riconosciuto
	surname	cognome dell'utente riconosciuto
	facebookid	Facebook ID dell'utente riconosciuto
	confidence	percentuale di precisione del riconoscimento

5.3 Screenshots applicazione nativa Android

Seguono alcuni screenshots della applicazione nativa, precendentemente sviluppata.





5.4 Sviluppo utilizzando il framework PhoneGap

In questo paragrafo viene fatta una panoramica sullo sviluppo e sulla implementazione della applicazione in esame. Nella prima parte viene descritta la logica in base alla quale l'applicazione è stata sviluppata utilizzando PhoneGap. Nella seconda parte si pone attenzione invece allo sviluppo grafico della applicazione. Nella terza parte vengono elencati i framework e le librerie esterne utilizzate, scendendo un pò più nel dettaglio ed esaminando come le

tecniche illustrate nei capitoli precedenti sono state integrate ed utilizzate nel progetto.

5.4.1 Logica di programmazione

La logica di programmazione che sta dietro ad ogni applicazione PhoneGap è molto semplice: una pagina HTML e uno script Javascript. Con questi soli due elementi è possibile realizzare una applicazione che sia platform-independent. La pagina HTML conterrà la struttura delle view da visualizzare, mentre invece lo script Javascript conterrà la logica dell'applicazione. Questa logica può sembrare banale, ed in effetti lo è. Risulta evidente che, soprattutto per applicazioni complesse, programmare utilizzando due soli file sorgenti risulta, oltre che sconveniente, sbagliato. Nello sviluppo dell'applicazione in esame si sono fatte le seguenti scelte:

- utilizzo di un solo file HTML (index.html), per quel che riguarda la parte grafica. Tale scelta viene motivata nel paragrafo successivo.
- organizzazione ed utilizzo di diversi file Javascript. Essendo la logica di programmazione fortemente event-driven, nel file HTML è stato incluso uno script (app.js) che ha il compito di registrare degli handler a degli eventi specifici, come ad esempio la pressione di un bottone o l'avvenuta acquisizione di una immagine.

Nel file HTML, vengono inoltre inclusi anche altri file e librerie Javascript, molte dei quali sono stati introdotti precedentemente.

L'applicazione in esame, dopo una più approfondita analisi dei requisiti, non richiede la presenza di una particolare configurazione di entità/oggetti. I requisiti evidenziano infatti come le funzioni principali dell'applicazione siano sostanzialmente le seguenti:

- l'interrogazione dei sensori fisici del device;
- l'interazione con server remoti;
- la presentazione dei dati ricevuti.

L'unica classe creata degna di nota e la classe User, che permette di costruire un oggetto che contiene le informazione dell'utente loggato.

5.4.2 Front-End Development

Nelle moderne applicazioni ha un ruolo decisamente importante la parte che si trova direttamente a contatto con l'utente e che, in molti casi, decreta l'interesse o meno dell'utente in una applicazione.

Nello sviluppo dell'applicazione in esame si è scelto di testare ed utilizzare JQueryMobile. Tale framework semplifica notevolmente lo sviluppo della parte Front-End e permette allo sviluppatore di concentrarsi di più sul funzionamento dell'applicazione. Per utilizzare il framework nel progetto le principali azioni da eseguire sono le seguenti:

1. includere la libreria Javascript e i fogli di stile;

```
<script type="text/javascript"
      src="jquery/jquery.js"></script>
<script type="text/javascript"
      src="jquery/jquery.mobile-1.0.1.js"></script
<link rel="stylesheet"
      href="jquery/themes/red.css" />
<link rel="stylesheet"
      href="jquery/jquery.mobile.structure-1.0.1.css"/>
```

2. strutturare la pagina HTML;

```
<body>
  <div data-role="primapagina">
      <div data-role="header">
      <h1>My Title</h1>
    </div><!-- /header -->
    <div data-role="content">
    <p>Hello world</p>
    </div><!-- /content -->
  </div><!-- /page -->

  <div data-role="altrapagina">
      <div data-role="header">
       <h1>My Title</h1>
    </div><!-- /header -->
```

```
    <div data-role="content">
    <p>Hello world</p>
    </div><!-- /content -->
  </div><!-- /page -->
</body>
```

Strutturando in questo modo il body della pagina HTML, JQueryMobile mostrerà all'utente la "primapagina". La struttura di ogni pagina deve essere dichiarata staticamente in questo file. Questo è il motivo per cui nell'implementazione di Visage tutta la parte grafica è stata dichiarata in un solo file HTML. La vera comodità di questo framework risiede nel fatto che mette a disposizione dei componenti e delle funzioni che semplificano la programmazione e fanno risparmiare tempo allo sviluppatore. Ad esempio, per avere una barra fissa nella parte alta della pagina è sufficiente creare un div con l'attributo "data-role" settato con la stringa "header". Lo stesso semplice meccanismo può essere usato per avere una barra fissa a fondo pagina, settando l'attributo con "footer" . Un altro punto a favore di questo framework è sicuramente il fatto che le transizioni tra view sono già implementate e sono del tutto simili a quelle native. Un ultima nota a favore è il fatto che JQueryMobile mette a disposizione dello sviluppatore strumenti per definire manualmente e visivamente dei temi grafici da applicare alle proprie applicazioni.

3. quando necessario, invocare le funzioni di libreria. Le funzioni di libreria sono molto semplici e permettono ad esempio di: effettuare chiamate AJAX; cambiare pagina facendo partire una transizione; altre funzioni di utilità.

JQueryMobile fornisce numerosi esempi e una documentazione molto esaustiva, per cui per ogni ulteriore approfondimento si rimanda al sito ufficiale, precedentemente indicato.

5.4.3 Implementazione

La programmazione con PhoneGap è del tutto analoga alla programmazione di applicazioni web con Javascript. L'unica vera differenza è che con PhoneGap le pagine HTML e gli script non devono essere scaricati dal browser

a seguito di richieste HTTP, ma sono già presenti "in locale". Ogni applicazione PhoneGap può ovviamente caricare contenuti "dinamici", attraverso semplici chiamate AJAX. Le tecniche di programmazione sono quindi analoghe a quelle relative ad applicazione web mobile, come illustrato nel capitolo 3. Passando al caso preso in esame, per lo sviluppo di Visage si sono dovute affrontare diverse problematiche più o meno complesse.

Inclusione di librerie

La prima questione fondamentale da risolvere è stata quella sul come includere le librerie di PhoneGap. PhoneGap presenta infatti file di libreria diversi per ogni piattaforma, ma che offrono sempre una stessa interfaccia, consistente per tutte le piattaforme. Siccome l'obbiettivo finale è mantenere tutti i sorgenti relativi alla parte HTML-CSS-JS insieme, senza dover ogni volta andare a cambiare a mano singoli file, la scelta più logica è stata quella di creare una sottocartella per ogni piattaforma (in questo caso due, iOS e Android). Tale problema è stato risolto utilizzando il feature detection e, più in particolare, è stato fatto uno sniffing dello useragent del browser, nel seguente modo:

```
var ua = navigator.userAgent;
var platform = {
iphone: ua.match(/(iPhone|iPod|iPad)/),
android: ua.match(/Android/)
};
if (platform.android) {
document.write('<script src=
    "js/android/phonegap-1.4.1.js"><\/script>');
} else if (platform.iphone) {
document.write('<script src=
  "js/iphone/phonegap-1.4.1.js"><\/script>');
}
```

In questo modo si possono ad esempio includere anche eventuali plugin esterni che, interfacciandosi a codice nativo, sono per forza di cose implementati in modo differente.

Integrazione con Facebook

L'integrazione con Facebook è un requisito fondamentale per questa applicazione. PhoneGap mette a disposizione un plugin che rende tale integrazione relativamente semplice. L'importazione di tale plugin consiste nel copiare all'interno dell'applicazione classi di codice nativo e modificare qualche file di configurazione. Tale plugin fornisce una interfaccia comune per cui, una volta seguite le procedure di importazione, è possibile interfacciarsi ai servizi di Facebook in maniera omogenea indipendentemente dalla piattaforma. Ulteriori informazione riguardo al plugin sono disponibili nella pagina seguente pagina GitHub: https://github.com/davejohnson/phonegap-plugin-facebook-connect.

Registrazione e Login

I meccanismi alla base dei processi di registrazione e login sono stati descritti in precedenza. La loro implementazione è molto semplice ed è formata dai seguenti step:

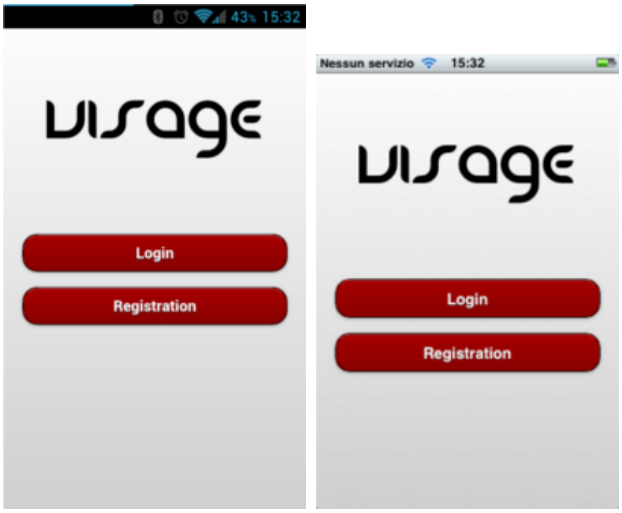
1. recupero dei dati tramite form;

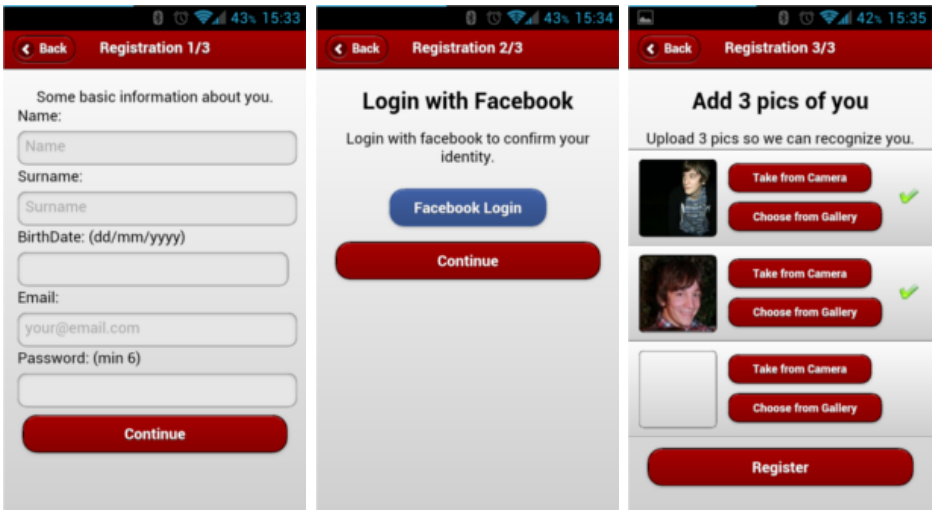
2. invio dei dati, tramite chiamate AJAX;

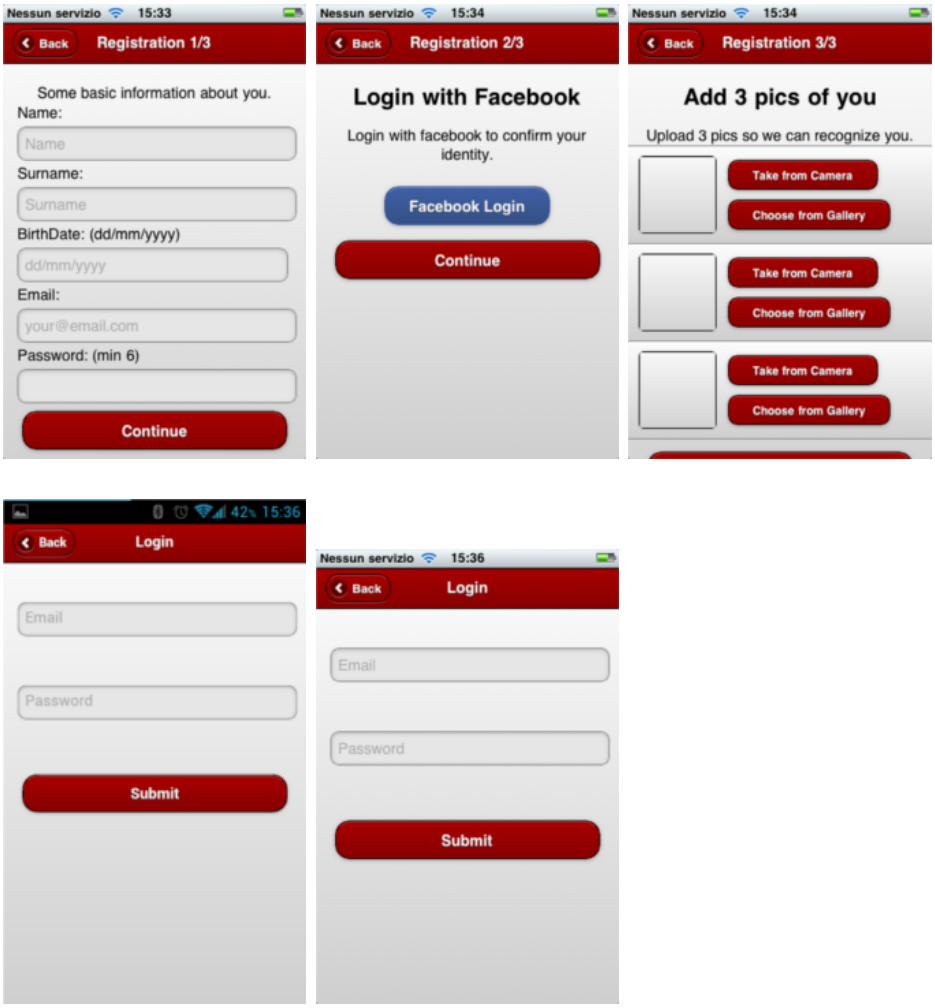
3. ricezione della risposta.

La permanenza dei dati, che permette ad esempio di ricordare l'utente loggato, è stata ottenuta utilizzando Local Storage di HTML5. Le richieste AJAX sono compilate e gestite utilizzando XUI. In seguito al login, si viene rediretti alla pagina del proprio profilo, che viene "riempita" dinamicamente, utilizzando il view templating e quindi, la libreria Mustache precedentemente introdotta.

Le seguenti immagini mostrano gli screenshots delle varie view di registrazione e login.


15:32
visage
Login
Registration
Nessun servizio
15:32
visage
Login
Registration


15:33
Back
Registration 1/3
Some basic information about you.
Name:
Name
Surname:
Surname
BirthDate: (dd/mm/yyyy)
Email:
your@email.com
Password: (min 6)
Continue
15:34
Back
Registration 2/3
Login with Facebook
Login with facebook to confirm your identity.
Facebook Login
Continue
15:35
Back
Registration 3/3
Add 3 pics of you
Upload 3 pics so we can recognize you.
Take from Camera
Choose from Gallery
Take from Camera
Choose from Gallery
Take from Camera
Choose from Gallery
Register



Riconoscimento

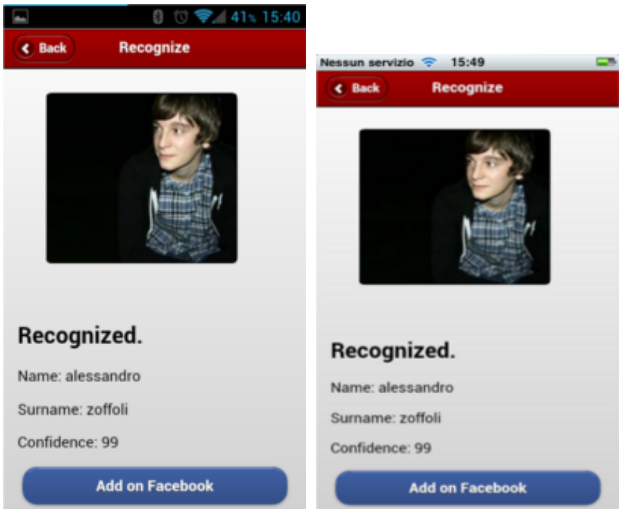
La funzione primaria delle applicazione è il riconoscimento di volti, e quindi, di persone. Tale funzione si traduce fondamentalmente in tre step:

1. acquisizione dell'immagine; L'acquisizione dell'immagine viene eseguita utilizzando le API di PhoneGap che riguardano l'accesso alla camera del dispositivo. Il loro funzionamento è stato precedentemente descritto. L'utente può scegliere se scattare una foto sul momento o se scegliere una immagine dalla propria galleria immagini.

2. invio dell'immagine, tramite chiamata AJAX; Tramite richiesta AJAX l'immagine viene mandata ai server, utilizzando le convenzioni illustrate precedentemente.

3. ricezione della risposta, contenente le informazioni riguardanti l'utente riconosciuto, o un errore.

In caso di esito positivo, il server risponde con una serie di informazioni relative all'utente riconosciuto. Tali informazioni vengono poi presentate in una nuova view, che contiene l'immagine dell'utente riconosciuto, alcuni suoi dati, ed un bottone che permette di aggiungere l'utente come amico su Facebook. Per costruire tale view viene nuovamente usato Mustache. Segue uno screenshoot.



Localizzazione

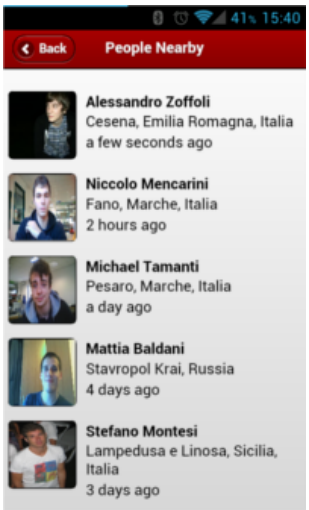
Il servizio di localizzazione è del tutto analogo ai precedenti come struttura logica. Gli step fondamentali sono:

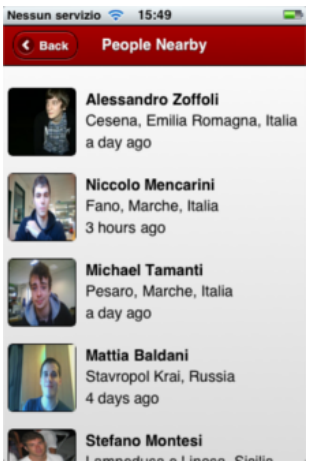
1. acquisizione delle coordinate GPS; L'acquisizione delle coordinate GPS dell'utente consiste nell'eseguire una chiamate alle API PhoneGap relative alla Geolocalizzazione, illustrate precedentemente.

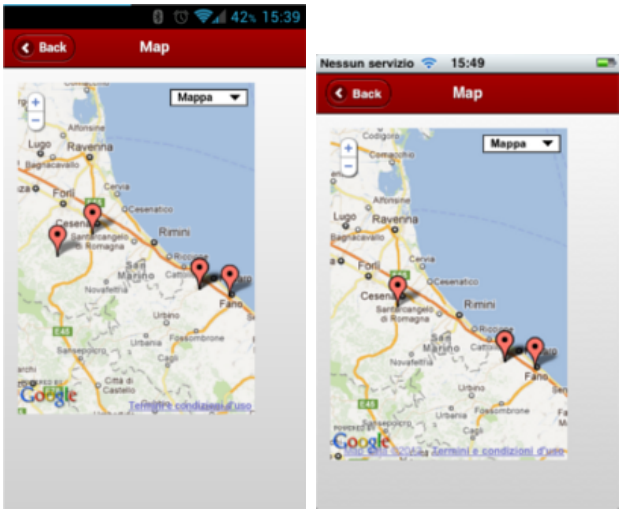
2. invio delle coordinate, tramite chiamata AJAX; Tramite richiesta AJAX le coordinate vengono mandate al server, che provvederà a memorizzarle ed utilizzarle opportunamente.

3. ricezione della risposta, o di un errore.

In caso di esito positivo della richiesta, il server provvede a restituire una lista di utenti posti nelle vicinanze (o meno) dell'utente. Tali utenti vengono elencati in una lista, generata dinamicamente utilizzando view templating (Mustache). Cliccando su un utente, verrà aperta una nuova view, contenente una mappa geografica, dove la posizione di ogni utente è segnalata da un marker. Per l'inclusione della mappa sono state utilizzate le API Javascript di GoogleMaps. Seguono alcuni screenshots.







5.5 Considerazioni finali

Come dimostrano gli screenshots del paragrafo precedente, l'obbiettivo di scrivere una applicazione platform-independent è stato raggiunto. Lo sviluppo dell'applicazione è stato semplice ma non senza problemi. PhoneGap è un ottimo framework, ma non è esente da piccoli bug.

L'unico obbiettivo che non è stato raggiunto nell'implementazione dell'applicazione è stato la realizzazione di un sistema di realtà aumentata. Il framework infatti non mette a disposizione delle API che consentano l'accesso diretto allo streaming delle immagini provenienti dalla fotocamera del dispositivo. Per realizzare tale sistema la soluzione più logica sarebbe quella di implementare un plugin, ovvero di andare a scrivere codice nativo platform-specific.

L'esperienza utente è gradevole e si presenta uniformemente su entrambe le due piattaforme target. Sicuramente grande merito va a JQueryMobile. Le prestazioni e la fluidità dell'applicazione sono in generale buone, anche se in certi momenti (in particolare durante certe transition CSS3) risultano leggermente inferiori rispetto alla controparte nativa.

Capitolo 6

Note critiche e sviluppi futuri

Nella prima parte di questo capitolo verranno elencate e descritte quelle che sono le maggiori criticità e mancanze che l'utilizzo di questi nuovi approcci e tecnologie comporta. Nella seconda parte viene invece lanciato lo sguardo verso il futuro, introducendo un nuovo possibile approccio alternativo, che si propone di colmare i limiti concettuali degli approcci attuali.

6.1 Note critiche

PhoneGap, utilizzato unitamente alle moderne tecnologie web, è uno strumento che può realmente semplificare la vita allo sviluppatore, ma una più approfondita analisi porta l'emergere di alcune questioni cruciali, analizzate in seguito.

6.1.1 Modello architetturale

Programmare applicazioni mobile utilizzando tecnologie legate al web semplifica certamente la vita allo sviluppatore e permette di ottenere un prodotto di una discreta qualità in tempi brevi. Passando ad un livello di astrazione maggiore è possibile però constatare come il modello che le applicazioni mobile realizzate con tali approcci utilizzano sia strettamente event-driven ed utilizzi uno stile continuation-passing.

Tutto il ciclo di vita dell'applicazione è sostanzialmente scandito da eventi. Si pensi ad esempio al fatto che utilizzando PhoneGap, per compiere

qualsiasi operazione bisogna attendere l'avvenuto caricamento del framework, il cui termine è scandito da un evento, che lo script che controlla l'applicazione deve intercettare. Tale script sostanzialmente deve avere il compito di registrare opportunamente gli handler agli eventi opportuni che regoleranno la logica dell'applicazione. Si pensi inoltre al fatto che ogni chiamata alle funzioni di libreria deve sempre comprendere un callback in caso di successo ed uno in caso di fallimento.

Tale approccio è sicuramente utile e veloce per quelle applicazioni che non sono troppo complesse, come può essere ad esempio una applicazione che che va ad interrogare una base dati e ne mostra i risultati. Tale approccio, che tende a progettare e scrivere codice basandosi sulle caratteristiche implementative del framework in uso, tende però a non tenere conto dei principi fondamentali dell'ingegneria del software. L'adempimento delle buone pratiche di progettazione e di programmazione lascia infatti il posto alla semplicità e alla velocità con cui lo sviluppatore riesce a concludere l'implementazione. Quando invece il livello di complessità aumenta, andando verso applicazioni più business-oriented, risulta essenziale l'applicazione di tali principi, pena il fallimento dell'intero processo produttivo del software[14].

Anche la programmazione nativa è sostanzialmente event-driven, ma non così strettamente come in PhoneGap. Infatti per ogni piattaforma il ciclo di vita una applicazione è scandito da eventi, cosi come l'intercettazione degli input provenienti dai sensori, ma i vari linguaggi di programmazione non pongono grossi limiti circa il modello architetturale da adottare.

6.1.2 Linguaggio

Un altro punto fondamentale su cui concentrarsi può essere anche dato dal linguaggio utilizzato per implementare la logica dell'applicazione. Tutti conoscono Javascript come il più famoso ed utilizzato linguaggio di scripting. La vera questione che emerge è quindi un'altra: é Javascript un linguaggio sufficientemente maturo, tale da poter sostenere progetti complessi e manutenibili nel tempo? I limiti di Javascript come linguaggio ad oggetti sono noti, e vanno dalla mancanza dei costrutti delle interfacce e delle classi astratte, all'avere un meccanismo di estensione non proprio elementare.

6.1.3 Multithreading

Sviluppare una applicazione utilizzando PhoneGap equivale sostanzialmente a sviluppare una web application, che verrà quindi eseguita in un browser. Nel caso di applicazioni PhoneGap, il browser è costituito da una web view. Il codice Javascript in un browser viene eseguito su un singolo thread, condiviso con il render della pagina. Risulta quindi evidente come una operazione di una certa complessità possa portare al freeze di una pagina. Javascript non consente inoltre di creare thread, ma soltanto di effettuare chiamate asincrone, che sono eseguite in un thread secondario, non gestito dallo sviluppatore, ma dal browser.

La mancanza del supporto al multithreading è una questione fondamentale che differenzia la programmazione nativa da quella che utilizza questi nuovi approcci e risulta difficilmente colmabile.

6.2 Futuro del mobile developing

Dopo aver analizzato ed esplorato le funzionalità di PhoneGap, è lecito chiedersi cosa verrà dopo PhoneGap, ovvero quali altre tecnologie potrebbero affermarsi e diffondersi nello sviluppo mobile. Per rispondere a questo interrogativo, si può partire dall'analisi di quello che è forse il difetto più grande di PhoneGap, ovvero lo stile architetturale, precedentemente introdotto.

Le tecnologie web utilizzate da questo framework sono state originariamente pensate e progettate per il recupero e la presentazione di informazioni. Tali operazioni non dovrebbero comprendere quindi operazioni computazionalmente complesse e l'approccio event-driven risulta nella maggior parte dei casi adeguato e sufficiente.

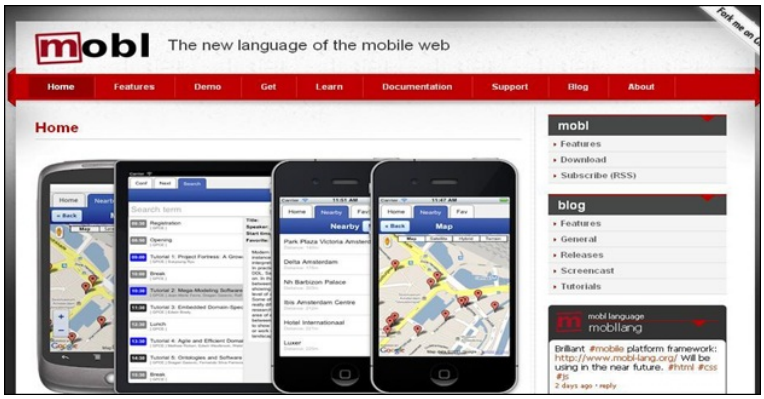
Nel caso di applicazioni che comportino una parte computazionale più o meno complessa, lo sviluppatore che utilizza queste tecnologie si trova nella situazione di dover frammentare il suo codice, spesso in maniera disordinata, in quanto la programmazione per browser forza le applicazioni ad essere scritte in continuation-passing style.

Bisogna prestare attenzione al fatto che non è che le tecnologie utilizzate siano inadeguate a realizzare operazioni che abbiano una certa complessità, ma è invece il modo in cui gli sviluppatori sono abituati ad utilizzare tali

tecnologie che le rende inadeguate. Ciò che deve cambiare è quindi il modello architetturale.

Risulta necessaria la realizzazione di una nuova ulteriore astrazione, che permetta di arrivare a modelli di sviluppo più adatti. Un ruolo fondamentale lo giocherà la ricerca, che dovrà studiare e costruire questa nuova astrazione, partendo dal presupposto che quella attuale risulta inadeguata.

6.2.1 Nella ricerca: Mobl



Un primo passo in questa direzione è stato fatto da Mobl[12], un nuovo linguaggio per il mobile web developing[15] che ha suscitato l'interesse di molti ricercatori. I creatori di Mobl partono da un presupposto fondamentale: le moderne tecnologie web permettono la creazione di applicazioni con un range di funzionalità sempre più ampio, a scapito però della development-experience, giudicata non sufficientemente consistente. I principali punti di partenza da loro elencati sono i seguenti:

- Polyglot programming: lo sviluppo di applicazioni con tecnologie web comporta l'utilizzo di diversi linguaggi domain-specific, come ad esempio HTML per la struttura, CSS per lo stile, Javascript per la logica. Ciò permette sicuramente di realizzare "separations of concerns", ma rende più difficile la risoluzione degli errori.

- Continuation-passing style: si riferisce sostanzialmente alla problematica principale emersa dall'utilizzo di PhoneGap, ovvero all'utilizzo di un modello architetturale event-driven con uno stile continuation-passing, che porta lo sviluppatore a rendere l'applicazione "un'insieme di frammenti di codice".

- User interface reuse: il problema fondamentale è dato dal fatto che HTML non offre nessun meccanismo di astrazione che permetta allo sviluppatore di riutilizzare frammenti di codice. Tale funzionalità può essere implementata solo tramite framework esterni, come visto in precedenza.

- Data binding: la questione da affrontare in questo punto riguarda il binding tra il modello dei dati e la rappresentazione di tali dati.

- Navigation: si riferisce alla natura della maggior parte delle applicazioni web/mobile, che consiste nel mostrare/nascondere una sequenza di view nello schermo del device, a seconda dell'interazione con l'utente.

I ricercatori responsabili del progetto hanno quindi creato un nuovo linguaggio di programmazione, con una sintassi molto simile a quella di Javascript, tipizzato staticamente, che mette a disposizione strumenti per definire interfaccie utente e modelli dei dati. Il compilatore traduce tale linguaggio in un insieme di file HTML e Javascript. Mobl si presenta quindi ottimamente agli occhi degli sviluppatori.

Mobl è un linguaggio per il web e non permette l'accesso ai dati provenienti dai sensori dei dispositivi su cui è in esecuzione. Una applicazione Mobl è di norma una normale web application, e deve quindi essere scaricata alla prima connessione, come un qualsiasi file HTML o script Javascript.

Combinando però l'utilizzo di Mobl con quello di PhoneGap, è possibile creare applicazioni platform-independent, che non sono necessariamente legate al web, scritte con un linguaggio che utilizza un modello architetturale dichiarativo.

Il lavoro che gli sviluppatori di Mobl e PhoneGap devono svolgere è sicuramente lungo ed impegnativo, ma ad oggi risulta sicuramente quello con una prospettiva di successo più promettente.

Mobl è un linguaggio integrato, in quanto combina data modeling, design di interfaccie grafiche, descrizione di servizi e scripting in un unico linguaggio. Le sue principali peculiarità, per cui si differenzia considerevolmente dai liguaggi preesistenti, sono le seguenti:

- automatic data persistance: le dichiarazioni di entità definiscono un modello persistente dei dati, grazie ad un database interno che viene installato nel mobile device, in maniera del tutto trasparente.

- reactive programming: consiste nel data binding introdotto precedentemente. In HTML e Javascript il modello dei dati è separato da quello della presentazione. Se l'utente aggiorna un valore da interfaccia grafica, la modifica di tale valore nella corrispondente base dati non è automatico, ma deve essere implementato tramite codice. Stesso ragionamento per l'operazione inversa. Mobl permette invece l'esecuzione di tale operazione in maniera trasparente allo sviluppatore. Tale tipo di programmazione è detta reactive programming e consiste nella creazione di una sorta di connessione tra l'interfaccia e il modello dei dati.

- continuation-passing style transform: lo stile continuation-passing è già stato introdotto in precedenza. Un esempio di come uno sviluppatore possa eseguire una chiamata AJAX d'esempio può essere il seguente:

```
getCurrentPosition(function(pos){
  DataProvider.getNearbyConferences(pos.lat, pos.long,
    function(nearbyConferencesJson){
      for (...) {
        ...
      }
    });
});
```

  In questo esempio vengono fatte due chiamate asincrone ed appare chiaramente come lo stile continuation-passing sia decisamente inadeguato nello sviluppo di applicazioni complesse. Nell'esempio che segue viene invece mostrato come ciò viene superato con Mobl.

```
var pos = getPosition ();
var nearbyConferencesJson =
    DataProvider.getNearbyConferences(pos.lat, pos.long);
foreach ( ... ) {
  ...
}
```

Mobl non elimina il continuation-passing style, ma semplicemente lo rende trasparente allo sviluppatore, che è portato a scrivere delle chiamate asincrone con una sintassi che richiama le chiamate sincrone. Ciò che viene fatto a livello di compilazione è la trasformazione di queste chiamate, scritte in maniera sincrona, in chiamate scritte in continuation-passing style, realizzando quindi un'astrazione. Il continuation-passing style non risulta però completamente scomparso dal linguaggio, ma il suo utilizzo viene semplicemente limitato al caso della reazione agli eventi provenienti dalle interfaccie grafiche. Un esempio di come questo stile viene ora utilizzato può essere il seguente:

```
button ( "Ok" , onclick ={
  ...
});
```

Capitolo 7

Conclusioni

Le motivazioni che hanno portato alla redazione di questa tesi erano principalmente legati alla curiosità e alla volontà di scoprire questi nuovi approcci e tecnologie legati al fiorente mondo mobile. Le principali domande a cui rispondere erano:

- l'approccio cross-platform mediante tecnologie web è realmente una soluzione valida?
- le prestazioni delle applicazioni sono paragonabili alle controparti native?
- l'esperienza utente che si crea con questi approcci si integra con le piattaforme su cui le applicazioni vengono eseguite?
- fino a dove ci si riesce a spingere con queste tecnologie?

Una risposta assoluta per ognuna di queste domande ovviamente non esiste e solo il tempo decreterà il successo o il fallimento di queste tecnologie. Nei seguenti paragrafi si cerca di dare un parere quanto più oggettivo possibile a questi quesiti, senza rispondervi direttamente, ma argomentando il discorso basandosi su alcuni punti chiave.

7.1 Prestazioni

Un punto molto importante per il successo di una applicazione è sicuramente quello che riguarda le prestazioni. Le prestazioni di una applicazione web

semplice si possono misurare ad esempio considerando il tempo che questa impiega per portare a termine una operazione, come ad esempio può essere il caricamento di una pagina. Per le applicazioni oggetto di questa tesi, ovvero mobile cross-platform realizzate con tecnologie web, tali operazioni si limitano al caricamento di una pagina, la cui struttura è in locale ed il cui contenuto può essere in remoto. Il tempo per portare a termine una operazione di questo tipo sarà quindi dato dalla somma del tempo di download dei dati remoti e il tempo che impiega il motore grafico a presentare la pagina.

Il tempo di esecuzione è sicuramente l'aspetto più importante relativo alla valutazione delle prestazioni di una applicazione. Nel caso di applicazioni la cui esecuzione si basa sulla interpretazione di codice, come nel caso di applicazioni basate su Javascript, maggiore sarà il volume del codice da interpretare, maggiore sarà il tempo di esecuzione. La soluzione più ovvia sarebbe quella di ridurre al minimo il volume del codice scritto, ma ciò comporta inevitabilmente ad una perdita di manutenibilità del codice stesso.

Nonostante i grossi miglioramenti di prestazioni che i browser mobile stanno portando nell'esecuzione di codice Javascript, le applicazioni mobile sviluppate con tecnologie web risultano e risulteranno sempre più lente rispetto a quelle native. La vera sfida che le varie piattaforme dovranno sostenere, nel caso in cui questi nuovi approcci prendano effettivamente piede e diventino rilevanti rispetto agli approcci nativi, sarà quindi quella di minimizzare tale differenza.

7.2 Eterogeneità delle piattaforme

L'eterogeneità delle piattaforme è un altro punto cruciale da tenere in considerazione al momento della scelta della tecnologia con cui implementare una applicazione mobile.

Piattaforme diverse sono intrinsecamente diverse tra loro e offrono user-experience che differiscono più o meno profondamente l'una dall'altra. Un esempio lampante può essere quello relativo ai tasti fisici che i device delle diverse piattaforme hanno. I device Android e Blackberry ad esempio sono dotati di diversi tasti fisici con cui interagire con le applicazioni, come ad

esempio il tasto indietro, opzioni e cerca. Tali tasti invece non sono presenti su device iOS.

Un'altra, e forse ancora più evidente differenza, è da ricercare nel fatto che ogni piattaforma ha le sue peculiarità grafiche, le sue librerie UI e soprattutto le sue linee guida relative al design di applicazioni. Gli utenti di una specifica piattaforma sono quindi abituati ad utilizzare applicazioni con un design integrato con la specifica piattaforma. Ad esempio si veda l'applicazione di Twitter, descritta in precedenza.

Tali risultati potrebbero certamente essere riprodotti anche combinando l'utilizzo di tecnologie web, come CSS3 e HTML5, che permettono di avere una resa grafica al pari di quella dei componenti nativi. Il maggiore punto critico di questa possibile soluzione è il fatto che se per ogni piattaforma si vuole emulare la grafica nativa, il vantaggio di avere un unico codice che vada bene per tutte le piattaforme si riduce drasticamente.

Un altro modo per offrire agli utenti una esperienza utente pari a quella nativa è l'utilizzo di plugin, che verrà trattato nel paragrafo seguente.

7.3 Estensibilità

Il concetto di estensibilità riferita a queste nuove tecnologie come, nel caso trattato in questa tesi, PhoneGap, riguarda principalmente due fattori:

- gli sviluppi futuri;
- la possibilità dell'utente di creare e aggiungere nuove funzionalità.

Nel primo punto elencato lo sforzo principale lo devono sostenere gli sviluppatori di PhoneGap. Il processo di inserimento di nuove API è un processo relativamente lento, in quanto dietro a ciò ci deve essere una fase non solo di pianificazione, ma soprattutto di testing delle nuove funzionalità su ogni piattaforma supportata. Prima di introdurre nuove API infatti le nuove funzionalità devono essere studiate a fondo, di modo da permettere agli sviluppatori di creare una interfaccia che sia il più generale possibile.

Nonostante tale processo sia relativamente complesso, non vi sono dubbi che, se tali tecnologie si diffonderanno in maniera considerevole e se i mezzi a disposizione degli sviluppatori saranno opportuni, tale processo potrebbe sicuramente essere completato con successo. Ciò fornirebbe agli utilizzatori

(ovvero gli sviluppatori di applicazioni) una serie di funzionalità aggiuntive che ad oggi sono disponibili solo nativamente.

Un esempio per meglio capire a cosa ci si riferisce può essere ricercato nel motivo per cui, nell'implementazione di Visage, si è dovuto rinunciare al servizio di realtà aumentata. Se in futuro PhoneGap introdurrà delle API di accesso allo stream proveniente dalla fotocamera, tale servizio potrebbe essere facilmente implementato. Altrimenti l'unica alternativa a disposizione dello sviluppatore consiste nel secondo punto elencato, ovvero di aggiungere nuove funzionalità al framework, creando dei plugin.

Sotto questo aspetto gli sviluppatori di PhoneGap hanno fatto un ottimo lavoro, creando un meccanismo semplice ed efficacie che permette agli sviluppatori di creare ed aggiungere nuove funzionalità.

La principale problematica nella creazione di plugin consiste nel rendere tali plugin compatibili con più piattaforme, ma la soluzione a questo è molto semplice. Basterà infatti creare una interfaccia omogenea, comune a tutte le diverse implementazioni, e le nuove funzionalità saranno viste come fossero API di PhoneGap.

Visto il crescente interesse verso questo framework, l'aggiunta di nuove funzionalità supportate ufficialmente nelle API di PhoneGap non può che incrementarne la facilità di utilizzo e la completezza.

Ringraziamenti

Desidero innanzitutto ringraziare il Professor Alessandro Ricci per avermi seguito nella stesura di questa tesi e le per le numerose ore che mi ha dedicato. Inoltre, ringrazio sentitamente i miei compagni di corso, in particolare Mattia Baldani, Michael Tamanti, Niccolò Mencarini e Stefano Montesi, sempre disponibili a collaborare e risolvere ogni mio dubbio. Infine, desidero ringraziare la mia famiglia, la mia ragazza, i miei amici e chi mi stà intorno, per essermi stati vicino durante questi tre anni.

Bibliografia

[1] Sito ufficiale di JQueryMobile, 2012.
http://jquerymobile.com/.

[2] Sito ufficiale di Sench Touch, 2012.
http://www.sencha.com/products/touch/.

[3] Sito ufficiale di The M Project, 2012.
http://the-m-project.net/.

[4] Sito ufficiale di PhoneGap, 2012.
http://phonegap.com/.

[5] Sito ufficiale di Jo, 2012.
http://joapp.com/.

[6] Sito ufficiale di DHTMLX Touch, 2012.
http://www.dhtmlx.com/touch/.

[7] Sito ufficiale di Titanium Mobile, 2012.
http://www.appcelerator.com/.

[8] Sito ufficiale di XUI, 2012.
http://xuijs.com/.

[9] Sito ufficiale di iScroll, 2012.
http://cubiq.org/iscroll-4.

[10] Sito ufficiale di Modernizr, 2012.
http://modernizr.com.

[11] Sito ufficiale di Mustache, 2012.
http://mustache.github.com/.

[12] Sito ufficiale di Mobl, 2012.
`http://www.mobl-lang.org/`.

[13] 2011, Andre Charland and Brian LeRoux.
Mobile Application Development: Web vs. Native.

[14] 2010, Anthony I. Wasserman.
Software Engineering Issues for Mobile Application Development.

[15] 2011, Zef Hemel and Eelco Visser.
Declaratively Programming the Mobile Web with Mobl.

[16] 2011, Andrew Lunny
PhoneGap Beginner's Guide. Packt Publishing.